Solving Large Airline Crew Scheduling Problems: Random Pairing Generation and Strong Branching

Diego Klabjan *
Ellis L. Johnson
George L. Nemhauser
School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0205

Eric Gelman
Srini Ramaswamy
Research and Development
Information Services Division
United Airlines

April 10, 2000

Abstract

The airline crew scheduling problem is the problem of assigning crew itineraries to flights. We develop a new approach for solving the problem that is based on enumerating hundreds of millions random pairings. The linear programming relaxation is solved first and then millions of columns with best reduced cost are selected for the integer program. The number of columns is further reduced by a linear programming based heuristic. Finally an integer solution is obtained with a commercial integer programming solver. The branching rule of the solver is enhanced with a combination of strong branching and a specialized branching rule. The algorithm produces solutions that are significantly better than ones found by current practice.

Keywords: transportation, branch-and-bound, airline crew scheduling

*Current address: Department of Mechanical and Industrial Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, 61801

1 Introduction

The airline crew scheduling problem concerns assigning crew itineraries to flights with the objective of minimizing the crew cost. A crew itinerary is called a pairing. The crew scheduling problem can be formulated as a set partitioning problem where flights correspond to ground set elements and pairings to subsets. The problem is difficult due to the large number of possible pairings, their complex structure, and nonlinear cost.

In this paper we present a new methodology for solving airline crew scheduling problems. First the LP relaxation of the set partitioning problem is solved to 'quasi' optimality. We solve it by repeatedly generating random pairings and reoptimizing. Overall we generate approximately half a billion pairings. For the integer programming phase, we select about 10 million pairings with low reduced cost. An integer solution is then found by a branch-and-bound heuristic algorithm. We enhance the solver with a new branching strategy. The overall crew scheduling algorithm yields better solutions than those produced by an algorithm currently used in practice. For some instances we were able to obtain solutions that are 3 times better. The algorithm was applied on a cluster of machines in parallel.

We summarize the contributions of the paper and how it is organized as follows. In Section 2 we present the new algorithm for airline crew scheduling. In Section 3 we give a detailed description of the random pairing generation routine. Connections are chosen randomly based on the connection times. A new branching rule, called the timeline branching rule, for airline crew scheduling is presented in Section 4. In Section 5 we show how to combine timeline branching with strong branching to enhance the branch-and-bound solver. We also experiment with a combination of follow-on and strong branching. The last section presents computational results including a comparison of different branching rules.

The Airline Crew Scheduling Problem

The input for an airline crew scheduling problem is a fleet together with its schedule and aircraft routing. Major US airlines operate based on a *hub and spoke* network. Airports where the activity is high are called *hubs* and the low activity airports, called *spokes*, are mostly served from hubs. This work focuses on domestic fleets of US airlines with a hub and spoke flight network.

A *flight leg* or *segment* is a nonstop flight. A *duty* is a working day of a crew consisting of a sequence of flights. A duty is subject to FAA and company rules. Among other rules, there is a minimum and maximum connection time between two consecutive flights in a duty, denoted by $minSit$ and $maxSit$. A connection within a duty is called a *sit connection*. The minimum sit connection time requirement can be violated only if the crew follows the plane turn, i.e. they do not change planes.

The cost of a duty is usually the maximum of three quantities: the flying time, a fraction of the elapsed time, and the duty minimum guaranteed pay. All three quantities are measured in minutes. We denote by dc_d the cost of a duty d.

Crew bases are designated stations where crews are based. A *pairing* is a sequence of duties, starting and ending at a crew base. A connection between two duties is called an *overnight connection* or *layover*. We refer to the time of a layover as the *rest*. Like sit connection times, there is a lower and an upper bound on the rest, denoted by $minRest$ and $maxRest$. If the rest period is longer than approximately 24 hours, we call it a *double overnight*.

The cost of a pairing is also the maximum of three quantities: the sum of the duty costs in the pairing, a fraction of the time away from base and a minimum guaranteed pay times the number of duties. The *excess cost* or *pay-and-credit* of a pairing is defined as the cost minus the flying time of the pairing. Note that the excess cost is always nonnegative. The *flight time credit* (FTC) of a pairing is the excess cost times 100 divided by the flying time.

A pairing is also subject to many FAA rules. A detailed discussion of legality rules and the cost structure can be found in Barnhart et al. [6]. For our input data the maximum number of segments in a duty is 10 and the maximum number of duties in a pairing is 4. Crews can also fly as passengers to be repositioned for the next flight, which is known as *deadheading*.

The *airline crew scheduling problem* is to find the minimum cost pairings that partition all the segments. Flight schedules generally repeat weekly. In the weekly airline crew scheduling problem we must find pairings that partition all the flight legs in the weekly schedule. Usually deadheads have to be considered. The daily airline crew scheduling problem is the crew scheduling problem with the assumption that each leg is flown every day of the week. In practice, some legs are not operated during weekends. Since the number of such irregular legs is small, the daily problem forms a good approximation to the weekly problem. The methodologies developed in this paper can be applied to both daily and weekly problems.

Traditionally a crew scheduling problem is modeled as the set partitioning problem

$$\min\{cx : Ax = 1, x \text{ binary}\}, \tag{1}$$

where each variable corresponds to a pairing, $a_{ij} = 1$ if leg i is in pairing j and 0 otherwise, and c_j is the pay-and-credit of pairing j. The number of pairings varies from 200,000 for small fleets, to about a billion for medium size fleets and to billions for large fleets.

Since crew cost is second only to fuel cost, the crew scheduling problem has drawn a lot of attention. In recent years, due to novel algorithmic methodologies and advances in computer hardware and software, the excess pay for large fleets has been considerably reduced. A recent survey is given by Barnhart et al. [6].

Modern airline crew scheduling branch-and-bound algorithms differ primarily on column selection since it is not possible to deal with all of the columns simultaneously. There are two main issues. One concerns when columns should be generated, with the two extremes being

- only at the root node
- at every node of the tree.

The other issue is how columns should be generated, with the two extremes being

- an optimization procedure that guarantees finding a column with lowest reduced cost
- random.

There are, of course, an infinite number of possibilities that can be thought of as convex combinations of these extreme alternatives.

When columns are generated throughout the tree, although not necessarily at every node, and an optimization approach, although not necessarily exact, is used to select columns, the algorithm is called *branch-and-price*. Theoretically branch-and-price offers the best hope of finding a solution that is close to optimum, but the column generation requires the solution of a constrained shortest path problem and can be both memory and time consuming. Moreover, its application requires significant customization of the IP solver and best reduced cost columns may improve the LP value but not the IP value. Barnhart et al. [5] give a survey of branch-and-price approaches. The first application of branch-and-price to airline crew scheduling appears to have been described in Desrosiers et al. [13]. Descriptions of column generation, branching, and search strategies for branch-and-price algorithms for crew scheduling are given by Vance et al. [24] and Anbil, Forrest and Pulleyblank [1].

It is very unlikely that pure random generation of columns could provide a reasonable set for the ensuing optimization. However, by mixing randomness with column selection based on tight connections, it could be possible to generate a very large number of columns very fast and then let the LP optimization at the root node select a subset of these columns for the IP. The IP selection would be based on reduced cost, feasibility and again, randomness. The IP would then be solved using a commercial IP solver. An approach along these lines, without randomness, was proposed by Chu, Gelman and Johnson [11]. We enhance this work by including randomness in column selection, new specialized branching and a new parallel LP solver. Earlier related work is given in Anbil, Johnson and Tanga [3] and Bixby et al. [10].

The tradeoff between branch-and-price and the type of approach we have just described is the cost of repeated generation of very judiciously selected columns versus a less exact but much faster per column up-front generation of a very large number of columns. There is unlikely to be a definitive answer on which approach is better since conclusions are likely to be problem and implementation dependent.

2 The New Algorithm for Airline Crew Scheduling

The algorithm has two main phases. The first phase solves the LP relaxation to 'quasi' optimality. Solving the LP relaxation to optimality may be unnecessary since the ultimate goal is to find a good integer solution. The second phase consists of heuristically finding an integer solution based on the dual information of the LP relaxation. The algorithm flow is given in Figure 1. Each step is described in the remaining part of this section.

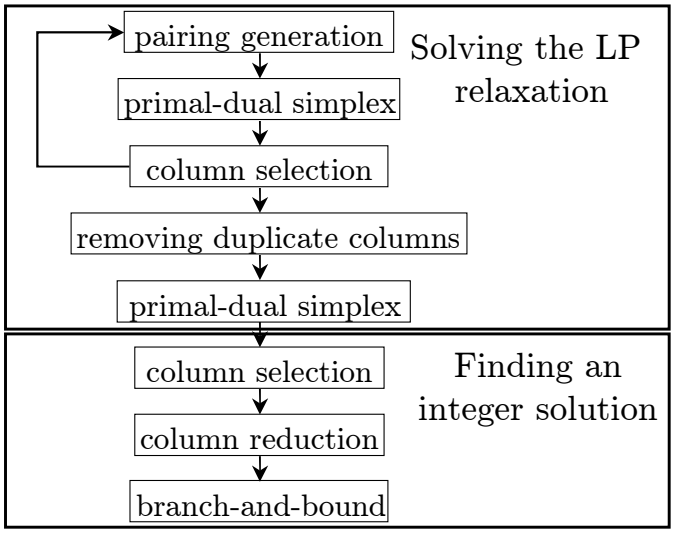

Figure 1: Algorithm flow

2.1 Solving the LP Relaxation

The LP relaxation is solved by repeatedly generating random pairings and then solving the resulting large-scale LP by a parallel primal-dual simplex algorithm. The motivation for the random pairing generation is that it is very fast and gives a good 'diversity' of pairings. The parallel primal-dual simplex is the fastest method we are aware of for solving these LPs with a small number of rows and many millions of columns. We explain the solution methodology for daily problems. The necessary modifications for weekly problems are given in Klabjan [17].

Let B be a primal feasible basis that is initially empty. At the end, B yields the best found primal feasible solution to the LP relaxation. We iterate the following steps (the loop in Figure 1).

Pairing generation: We generate between 50 and 75 million random pairings. The random aspect of the generation is described in Section 3. The pairings are then uniformly at random redistributed among all processors.

Primal-dual simplex: The LP with columns corresponding to generated pairings and columns that are in the basis B is solved by the parallel primal-dual algorithm of Klabjan, Johnson and Nemhauser [18].

Column selection: Next we select a set of pairings that are candidates for the IP. We store some low reduced cost pairings, approximately 2 million, and one million random pairings. After the low reduced cost pairings have been selected, the random pairings are chosen from the remaining ones with probability $\exp(-\tau \cdot rc^2)$, where rc stands for the reduced cost of the pairing and τ controls the number of selected pairings. The basis B becomes the optimal basis from the LP.

Since we keep the basis, the objective value cannot increase at any iteration. We break the loop when the decrease in the objective value falls below a given threshold.

For medium and large size fleets the number of iterations ranges from 10 to 15. Typically, initial decreases in the objective value are around 500 minutes and gradually they drop down to 20. When a decrease falls below 20 minutes, we stop. Note that in pairing generation we do not use any dual information. However, the process might be speeded up by pruning the depth-first search if the reduced cost of a partial pairing becomes 'too big'.

With k loops in this phase, we have selected approximately $3k$ million pairings. Next we first remove all duplicate columns by using the parallel algorithm described in Klabjan [17]. Then we solve the LP over the selected pairings (second 'primal-dual simplex' step in Figure 1). Note that we may improve the solution obtained in the previous phase.

2.2 Finding an Integer Solution

A common procedure for obtaining an IP is to select a set of columns with low reduced cost with respect to the dual vector of the LP relaxation. An integer solution is then found by a branch-and-bound algorithm using the selected columns, see Chu, Gelman and Johnson [11] and Anbil, Johnson and Tanga [3]. The number of selected columns in their work is between 10,000 and 15,000. Our experiments have shown that this is a good strategy for small fleets or whenever the reduced costs of columns are relatively high, e.g. weekly problems. Due to the high number of low reduced cost pairings in our instances, we select several million pairings with low reduced cost.

Based on the computed dual vector in the LP programming phase we select pairings with reduced cost below a given threshold (second 'column selection' step in Figure 1).

For our instances the threshold used was 50 or 100. Typically around 10 million pairings are chosen.

Next we use a heuristic to reduce further the number of columns to 100,000. The heuristic is based on the *follow-on* branching rule, Desrosiers et al. [13] and Anbil, Johnson and Tanga [3], that is motived by the Ryan-Foster branching rule, Ryan and Foster [23]. Consider two flight legs r and s. On one branch, called the follow-on branch, we force the two legs to appear consecutively in a pairing. On the other branch, called the non follow-on branch, the two legs can not appear consecutively. Denote by P_r the set of all pairings covering leg r and by P_{rs} the set of all pairings that contain both legs r and s with leg s immediately following leg r. It has been shown in Vance et al. [24] that follow-on branching is a valid branching rule. Namely, if x^* is an optimal fractional basic solution to the LP relaxation of (1), then there exist two flights r and s such that

$$0 < \sum_{i \in P_{rs}} x_i^* < 1 \, .$$

We can then form the two branches as follows. We fix the follow-on by setting all the variables corresponding to pairings in $(P_r \cup P_s) - P_{rs}$ to 0. In the non follow-on branch, all the variables corresponding to pairings in P_{rs} are set to 0.

The following procedure, called the *column reduction heuristic*, is iterated as long as the number of pairings stays above 100,000. Suppose that x^* is the current LP value. We fix all follow-ons (r, s) with $\sum_{p \in P_{rs}} x_p^* = 1$ and also the follow-on $(\tilde{r}, \tilde{s})$ with the biggest value of $\sum_{p \in P_{\tilde{r}\tilde{s}}} x_p^*$, but still less than 1. Note that the $(\tilde{r}, \tilde{s})$ follow-on cuts off the current fractional LP solution. Finally the resulting new LP is solved with a primal-dual algorithm and the procedure is repeated. Each fixed follow-on reduces the number of rows in the set partitioning problem by 1.

We apply the column reduction heuristic only if there are many low reduced cost pairings. However, if the number of such pairings is small, then we take 100,000 columns with the smallest reduced cost. Finally, we find an integer solution by using a commercial mixed integer programming solver that is enhanced with a strong branching rule described in Section 5.

3 Pairing Generation

The current practice of generating only a subset of pairings is either by generating pairings that cover specified subsets of legs, Gershkoff [16], Anbil et al. [2], or by prior knowledge of 'good' connections, Andersson et al. [4]. The former methodology of generating pairings is used in TRIP algorithms. The latter approach is a greedy one, i.e. only a given number of short connections are chosen.

Our new approach combines the greedy estimates based on connection times with randomization. Such an approach is called the *greedy randomized adaptive search procedure* (GRASP), see e.g. Feo and Resende [15].

There are two networks used for pairing generation, Barnhart et al. [6]. The *segment timeline network* has two distinct nodes for each flight, one for the arrival and the other for the departure. For each flight there is an arc connecting the two nodes. Additionally the network has an arc between the arrival node of a flight and the departure node of a flight if the connection time between the two flights is shorter than $maxSit$ and the arrival station of the first flight is the same as the departure station of the second flight. The *duty timeline network* on a given set of duties is defined in a similar way except that connection times are required to be within $[minRest, maxRest]$.

Each duty is a path in the segment timeline network and each pairing is a path in the duty timeline network. However due to pairing and duty feasibility rules a path is not necessarily a pairing or a duty. Our pairing generation is based on a duty timeline network. First duties are generated from the segment timeline network and then pairings are obtained from the duty timeline network. Throughout the computation we work only with a small subset of duties so memory requirements are not a problem, which is usually the difficulty with duty networks. For daily problems we employ the following strategy. Suppose that the maximum number of allowed duties in a pairing is d. We assume that we are not going to consider any pairing that has two double overnights due to the high cost. Hence any considered pairing cannot exceed $d+1$ days. Based on this assumption we add $2(d+1)$ nodes for each leg to the segment timeline network, each one corresponding to a different consecutive day of the week. Random duties are then constructed from such a network. Using this approach we generate different duties on different days of the week and hence we get a larger sample of duties than by generating them on a single day and making copies for other days of the week.

Duties and pairings are generated by using a depth-first search enumeration on the segment and duty timeline networks respectively. We attempt to extend a partial duty/pairing with a segment/duty if there is a corresponding connection arc in the network.

As already indicated above we first generate random duties and then random pairings. The two parts differ due to the number of possible connections even though the basic idea is the same. A segment typically has no more than 30 connections whereas a duty can have hundreds of connections.

3.1 Random Duty Generation

To generate random duties we choose random connections in the depth-first search procedure. Let t_{ij} be the connection time between flights i and j in the segment timeline network. Let $p_{ij} = f(t_{ij})$ be the probability of choosing the connection. The

depth-first search procedure attempts to extend the current duty with the connection arc (i, j), i.e. with the flight leg j, with probability p_{ij}. Note that first the connection is chosen and then the feasibility of the new duty is checked.

The main issue in the above procedure is the computation of probabilities. At spokes the connections are sparse and hence we set $p_{ij} = 1$ if the arrival station of leg i is a spoke. We use a parameter denoted by E for the expected number of selected connections at each node of the network. The parameter controls the number of random duties that are generated. The bigger the value, the more duties we generate. If a node has fewer than E connections, then all the connections are considered. Connections corresponding to plane turns that have connection time shorter than the minimum sit connection time are always selected.

For the remaining nodes we have to choose an appropriate function f. The function is nonincreasing since shorter connections are preferable, and it is convenient for it to be continuous. By using the exponential function

$$p_{ij} = e^{-\zeta_i (t_{ij} - minSit)^2} ,$$

where ζ_i is a parameter depending only on leg i, we assign considerable weight to the connection time. We have to compute the value ζ_i that satisfies the expected number of connections requirement, namely

$$g(\zeta) = \sum_j e^{-\zeta_i (t_{ij} - minSit)^2} = E \, . \tag{2}$$

By introducing $\xi_i = exp(-\zeta_i)$, the solution to the equation (2) is a root of a polynomial. To solve it, we use Newton's method with a starting point at 1 (see e.g. Bertsekas [8]). Since the number of considered connections is low, the method is fast. In our implementation we first compute ζ_i satisfying (2) for each node in the segment timeline network and then we carry out the depth-first search enumeration of duties.

3.2 Random Pairing Generation

Once random duties have been generated as described above, we generate random pairings. Typically we generate from 30,000 to 60,000 duties. Since too many pairings can be constructed from them, we generate only a subset of pairings.

The basic idea of generating pairings is similar to the duty generation; we want to expand a partial pairing with a duty by choosing connections with a certain probability. As above, a desired property of the probability is that longer connections should have a lower probability. However the straightforward adaption of the duty approach would be intractable due to a different order of magnitude in the number of nodes of the segment timeline network and the duty timeline network. For example, for each duty

to compute the value of ξ would involve finding a root of a polynomial of degree equal to the maximum rest time and with up to 500 (the number of possible connections of a duty) nonzero coefficients.

Let S_j be the set (cluster) of all duties that start with leg j. Denote by a_j the common departure time of all such duties. Assume that we want to extend a partial pairing that ends with a duty d. The duties we consider are all the duties that depart at the same station as the arrival station of the duty d and that satisfy the minimum and maximum rest time restrictions.

To circumvent the problem of computing the value of ζ for each duty, we first choose a random step size in time and then random duties from the first cluster of duties with the departure time after the randomly generated time. More precisely, if at the previous step we have been generating duties from the cluster S_j, we sample a random number of minutes denoted by n and in the next step we choose duties at random from the first duty cluster S_k whose departure time is greater than $a_j + n$, see Figure 2. We call the random variable n a step size. We use the normal distribution for the step size, namely $n \sim N(\mu, \sigma^2)$. The variance σ is fixed throughout the computation but the mean value μ varies based on the connection time. Suppose that t_j is the connection time between a duty in S_j and the duty d. Then $\mu = f(t_j)$, where f is an increasing, continuous function. We use a linear function for f. Once the next cluster of duties S_k is chosen, we need to generate random duties from the cluster. Given a fixed number τ, the probability of choosing a duty from S_k is $p = \exp(-\tau(t_k - minRest)^2)$.

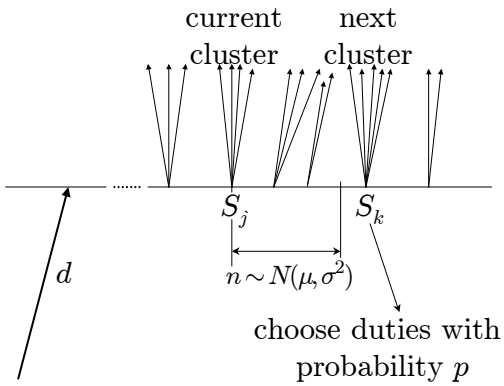

Figure 2: Selecting the next duty cluster

We now consider generating random duties from a cluster. Note that the probability p of choosing a duty depends only on the cluster and not on a single duty within the cluster. We use a parameter $pMethod$ where if $p > pMethod$, then we loop through all the duties in the cluster and we select a duty with probability p (binomial sampling). If $p \leq pMethod$, then we use geometric distribution sampling as follows. We select a

number u from the uniform distribution on $[0, 1]$ and we compute $\tilde{u} = \lfloor \ln u / \ln(1-p) \rfloor$ which is geometrically distributed with parameter p (see Law and Kelton [20]). We skip the next $\tilde{u} - 1$ duties and we select the duty that follows, see Figure 3. The procedure is repeated until all the duties in the cluster are scanned. Additional implementation details can be found in Klabjan [17].

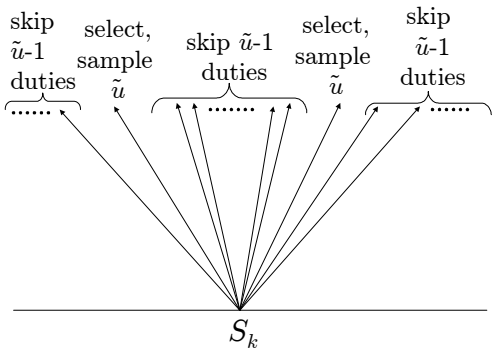

Figure 3: Selecting duties from a cluster: $p \leq pMethod$

As in the duty generation, if the number of all possible overnight connections is less than a given number, we do not apply the random scheme. For example, at a spoke the number of connections is small so we consider all of them for a possible extention of the partial pairing.

The random pairing generation routine was embedded into the parallel generation algorithm developed in Klabjan and Schwan [19]. The amount of additional work of a random pairing generation step is negligible if the pseudo-random number generator is fast. We report computational results in Section 6.3.

3.3 Generating Low FTC Pairings

Since it is unlikely that a good solution has pairings with really large FTC, we only generate pairings that have FTC below a given number K. In the depth-first search procedure we prune all partial pairings that would in the best possible scenario yield a pairing with an FTC bigger than K. The following proposition gives a lower bound on the FTC of a pairing.

Proposition 1. *Denote the maximum number of allowed duties in a pairing by maxDuties, the maximum allowed flying time in a duty by maxFly, and the flying time of a duty*

d by fl_d. Let a partial pairing have duties $d_1, \ldots, d_k$, where $k \leq maxDuties$. If

$$\frac{\sum_{i=1}^{k} dc_{d_i} + (maxDuties - k) \cdot maxFly}{\sum_{i=1}^{k} fl_{d_i} + (maxDuties - k) \cdot maxFly} \geq K + 1 \,,$$

where $0 < K < 1$ is a real number, then the FTC of any pairing resulting from the partial pairing is greater than K.

Proof. Assume that the partial pairing is completed to a pairing by appending the duties $d_{k+1}, \ldots, d_{k'}$. Then a lower bound on the cost of the pairing is $\sum_{i=1}^{k'} dc_{d_i} \geq \sum_{i=1}^{k} dc_{d_i} + \sum_{i=k+1}^{k'} fl_{d_i}$ since the cost of a pairing is bigger than the sum of the duty costs and $dc_{d_i} \geq fl_{d_i}$. Hence a lower bound on the ratio of the cost of the pairing and the flying time is

$$\frac{\sum_{i=1}^{k} dc_{d_i} + \sum_{i=k+1}^{k'} fl_{d_i}}{\sum_{i=1}^{k'} fl_{d_i}} \geq \frac{\sum_{i=1}^{k} dc_{d_i} + (maxDuties - k) \cdot maxFly}{\sum_{i=1}^{k} fl_{d_i} + (maxDuties - k) \cdot maxFly} \,,$$

since $fl_{d_i} \leq maxFly$ and $k' \leq maxDuties$. The bound can be checked by multiplying the fractions to eliminate denominators and expanding the products. The claim now easily follows. □

For daily problems we set the cutoff value $K = 0.25$. This additional pruning reduces the pairing generation time by a third.

4 Timeline Branching

The choice of an effective branching rule can be crucial to the solving of large-scale integer programs. Here we present a new branching rule called *timeline branching* for solving the set partitioning problem (1) by an LP based branch-and-bound algorithm. The idea of timeline branching comes from SOS branching, Beale and Tomlin [7]. Let $\tilde{P}_r = \cup_s P_{rs}$. The pairings in $\tilde{P}_r$ can be ordered based on the connection time with flight leg r. For each pairing $p \in P_{rs}$, we define the connection time, denoted by t_p, to be the departure time of the leg s minus the arrival time of the leg r. The timeline branching rule first identifies a leg r and a time t. The pairings in $\tilde{P}_r$ are then split based on the connection time and the time t. The first branch, called the 0 timeline branch, sets all the pairings $p \in \tilde{P}_r$ with connection time $t_p \leq t$ to 0; the second branch, called the 1 timeline branch, sets all the pairings $p \in \tilde{P}_r$ with connection time $t_p > t$ and the pairings that end with r to 0. Thus the 0 timeline branch can be expressed as

$$\sum_{p \in \tilde{P}_r, t_p \leq t} x_p = 0 \,,$$

and the 1 timeline branch as

$$\sum_{p \in \tilde{P}_r, t_p \leq t} x_p = 1 \, .$$

If there are no flights departing at the same time from the same station, then timeline branching is a valid branching rule.

Proposition 2. *Suppose that any two flights departing from the same station have different departure times. If x^* is a basic fractional solution to the LP relaxation of (1), then we can identify a leg r and a time t such that*

$$0 < \sum_{p \in \tilde{P}_r, t_p \leq t} x_p < 1 \, .$$

Proof. It is proved in Vance et al. [24] that under the conditions stated in the proposition, there are flights r and s such that $0 < \sum_{p \in P_{rs}} x^*_p < 1$. The identified leg for the timeline branching rule is r. If $\sum_{p \in \tilde{P}_r} x^*_p < 1$, then we can choose the time t to be the maximum rest time.

Now suppose that $\sum_{p \in \tilde{P}_r} x^*_p = 1$. Then there is a leg $q, q \neq s$, such that $0 < \sum_{p \in P_{rq}} x^*_p < 1$. By the assumption the departure times of legs s and q are different. We can then identify pairings $p_1 \in P_{rs}$ and $p_2 \in P_{rq}$ such that $0 < x^*_{p_1} < 1$ and $0 < x^*_{p_2} < 1$. Then we can set $t = (t_{p_1} + t_{p_2})/2$, which is different from t_{p_1} and t_{p_2}. □

If there are legs with equal departure time and departure station, we can slightly perturb the departure and arrival times. For our input data, 15% of the legs needed perturbation on average.

With an appropriate choice of the pair (r, t), the two branches can produce balanced branch-and-bound trees, even more balanced than those obtained with the follow-on branching rule. As with the follow-on branching rule, care has to be taken in choosing a good branching pair.

5 Strong Branching

Given a branching rule such as the conventional variable dichotomy rule, follow-on or timeline, the next decision is to choose a specific branching that meets the rule. Strong branching is a method for choosing a specific branching that achieves the objectives of either increasing the lower bounds on the child nodes as much as possible or finding a good integral solution quickly, or both. Assume that we use variable dichotomy as a branching strategy and we have a subset S of fractional variables that are candidates for branching. For each variable $i \in S$ the two branches are formed and a given number k of dual simplex iterations are performed on each branch. For $i \in S$ let f^0_i, f^1_i be the

resulting objective values. These values are used to choose a branching variable. If the two values for a variable i are relatively large, then i is a good candidate to branch on since there are significant increases in the LP value for both branches. There are three decisions in the strong branching rule that need to be specified: the choice of the set S, the number of dual iterations k, and how to combine the values f^0, f^1 to obtain a branching variable.

Strong branching first appeared in the commercial mixed integer programming solver CPLEX, CPLEX Optimization [12]. No details are known about the implementation. Bixby et al. [9] choose the subset S as the set of 10 least integral variables. The integrality of a variable i with the value x_i^* in an LP solution of the current node is defined as $|x_i^* - 0.5|$. The number of dual simplex iterations is 50. A variable maximizing $10 \max\{f_i^0, f_i^1\} + \min\{f_i^0, f_i^1\}$ is selected as the branching variable. A slightly different approach is presented in Linderoth and Savelsbergh [21]. The set S is chosen as the set of variables having the LP value between a given lower and upper bound. The number of dual iterations used is 25 and the branching variable is a variable maximizing $f_i^0 + f_i^1$.

We generalize the strong branching ideas to capture our branching rules. The values f^1 in the follow-on branching strategy correspond to the objective values of the follow-on branches after carrying out a given number of dual simplex iterations. The f^1 values for timeline branching correspond to the 1 timeline branches. The f^0 values correspond to the other branches in both branching rules.

We first focus on strong branching with the follow-on rule, called *strong follow-on branching*. The subset S is chosen as a set of 180 least integral follow-ons. The integrality of a follow-on (r, s) is defined as

$$\left| \sum_{p \in P_{rs}} x_p^* - 0.5 \right| .$$

Since there are typically fewer fractional variables in variable dichotomy than there are fractional follow-ons, a larger size set S is reasonable. For each follow-on in S, we perform 20 dual simplex iterations. We choose the branching follow-on by $\max_{i \in S}\{f_i^0 + \alpha f_i^1\}$, where α is a parameter. Since the follow-on branch is a branch revealing more information and more likely to yield an integer solution soon, we would like to have a large lower bound improvement in the non follow-on branch and a small lower bound improvement in the follow-on branch. Note that a large lower bound improvement means that it is likely to prune the branch and a small improvement translates into high chance of finding a good integral solution. Therefore it makes sense to require $\alpha \leq 1$. We experimented with the settings $\alpha = 0.8, 0.5, -1$ and found out that the value -1 outperforms the other two for all tested instances. Hence the branching

follow-on is the one attaining the maximum in

$$\max_{i \in S} \{ f_i^0 - f_i^1 \} .$$

Strong branching with the timeline rule needed a little bit more experimenting and a slight divergence from standard strong branching. The integrality of a branching pair (r, t) is defined as

$$| \sum_{p \in \tilde{P}_r, t_p \leq t} x_p^* - 0.5| .$$

Since a 'high' level branching decision should be made at the top of the tree, we choose to make decisions of the connection type, i.e. sit connection or overnight connection, first. Hence the only timeline branching pairs considered are of the form $(r, maxSit)$. We call this *sit/layover branching*. Note that fractional solutions cannot always be excluded by sit/layover branching since there may not exist a time t that divides the timeline as in sit/layover and satisfies the conditions of Proposition 2. Therefore when sit/layover cannot be executed, it must be replaced by a generally applicable branching rule such as variable dichotomy.

We perform strong branching in conjunction with sit/layover branching if there is at least one pair $(r, maxSit)$ with integrality less than 0.1. The candidate set S is the set of 20 least integral timeline branching pairs and 20 dual simplex iterations are performed. Since in timeline branching neither branch is clearly better than the other, the branching pair in strong branching is one that satisfies

$$\max_{i \in S} \{ f_i^0 + f_i^1 \} .$$

When there is no pair with integrality less than 0.1, we switch to variable dichotomy branching, specifically branching on the most fractional pairing. We call this rule the *strong sit/layover branching rule*. An alternative is to combine sit/layover branching with either strong variable dichotomy branching or strong follow-on branching. However, since after making sit/layover decisions on the top of the branch-and-bound tree, the number of still active pairings is small, we decided to combine sit/layover with conventional variable dichotomy as indicated above.

6 Computational Results

6.1 Parallel Computing Environment

All computational experiments are performed on clusters of machines. Two clusters are used, the first consisting of 16 200MHz Quad Pentium Pros and the second comprised of 48 300MHz Dual Pentium IIs, resulting in 160 processors available for parallel program

execution. All machines are linked via 100 MB point-to-point Fast Ethernet switched via a Cisco 5500 network switch. Each machine with a Quad Pentium has 256MBytes of main memory whereas the remaining 48 nodes have 512MBytes of main memory per machine.

In summary, the cluster machine we use is representative of typical machines of this type. It has a relatively slow internode communication but has a good cost/performance ratio in comparison with specialized parallel machines like the CM-5, the Intel Paragon, or the IBM SP-2 machines. The Intel cluster machine also shares characteristics with modern parallel machines like the IBM SP-3, in its use of multiprocessor nodes, with 8 processors/node used in the IBM SP-3 vs. the 4 processors used in our Intel cluster.

The parallel implementation uses the MPI message passing interface (see e.g. Message Passing Interface Forum [22]), MPICH implementation version 1.0, developed at Argonne National Labs. The MPI message passing standard is widely used in the parallel computing community. It offers facilities for creating parallel programs to run across cluster machines and for exchanging information between different processors using message passing procedures like broadcast, send, receive and others.

The mixed integer programming solver used was CPLEX, CPLEX Optimization [12], version 5.0.

6.2 Computational Results with Branching

All computational experiments in this section were performed on the cluster consisting of 16 200MHz Quad Pentium Pros. We embedded the branching rules within the mixed integer programming solver CPLEX. In all experiments a steepest edge dual simplex algorithm was used for solving the LP relaxations. We tested our branching rules against the solver's default setting. We tried to tune some parameters of the solver but the default setting produced the best results.

We first performed computational experiments with various node selection options and without the use of strong branching, i.e. only standard follow-on or timeline branching was used. The solver's default branching rule constantly outperformed both follow-on and timeline branching rules. This is a clear indication that strong branching is needed. For the remaining experiments we fixed the node selection strategy to the best bound node and all other features of the solver were left unchanged.

A comparison of the CPU time of the solver with an implementation of the strong branching rule and with the default branching rule is hard due to the following two facts. The current implementation of the solver first carries out the default branching rule and then the user defined branching function is called. Hence there is an additional overhead of performing the default branching rule. The second argument relies on the fact that the internal data structures of the solver are not available in a user-defined branching function. For example, the same tableau has to be computed for each child

node before performing the dual iterations in strong branching.

The strong sit/layover branching rule is not much more computationally intensive than the default branching rule because strong branching decisions are performed only at a few nodes at the top of the branch-and-bound tree. Therefore the limits on the number of evaluated nodes for the strong sit/layover branching rule and the solver's default branching rule were the same (about 12 hours). However, the node limit for the strong follow-on branching rule, which is much more computationally intensive than the default, was determined in such a way that the execution times were roughly the same as for the default.

Note that the strong branching rules are supposed to work well on big mixed integer instances where solving LP relaxations is relatively time intensive. We found out that it pays to use strong branching for problems with more than 150 rows assuming that the number of columns is larger than 50,000.

To reduce the overall computational time, the strong branching rule is performed in parallel. Each processor has a copy of the preprocessed formulation. At the beginning of a branching procedure, the upper bounds at the current node are broadcasted to each processor. In addition the basis and the dual norms associated with the node are broadcasted to all processors. A dedicated processor computes the follow-ons that form the set S and then distributes them evenly to all other processors. The processors then in parallel perform the dual simplex iterations on both child nodes and for all assigned follow-ons. At the end the best follow-on from all processors is chosen.

The computational results are presented in Table 1. The number of columns and rows reported is the number after the preprocessing step has been carried out by the solver. All the instances are daily problems. The "Num. nodes" column reports the total number of evaluated nodes. In runs denoted by * an optimal solution was found. We see that strong branching rules outperform the default solver. The strong sit/layover branching rule found the overall best solution in the first 2 instances, but the CPU times for finding the best integer solution were longer than the corresponding times for strong follow-on branching.

We performed additional experiments with strong follow-on branching on two weekly instances. The computational results are shown in Table 2. Strong follow-on branching clearly outperforms the default solver's branching rule. In these runs, we did not consider strong sit/layover branching since the results in Table 1 indicate that it takes more time to find a good integer solution.

6.3 Computational Results with the Crew Scheduling Methodology

Here we discuss solution quality. The instances and the FTC results are summarized in Table 3. The problems $fl2$ and $fl3$ were used in Vance et al. [24]. The feasibility rules and the cost function used are identical to those in [24].

Num. rows	Num. cols	Branching rule	Solution value	Num. nodes	Node index best solution
179	99150	default	1,380	10,000	200
		strong follow-on	1,215	600	53
		strong sit/layover	1,052	10,000	9,716
207	95605	default	344	10,000	559
		strong follow-on	209	700	104
		strong sit/layover	166	10,000	3,005
211	70422	default	436	10,000	9,829
		strong follow-on *	420	2,651	520
		strong sit/layover *	420	10,000	7,213
178	195647	default	∞	3,000	-
		strong follow-on *	55	589	404
		strong sit/layover *	55	1,970	1,806
244	98179	default	∞	6,800	-
		strong follow-on	631	500	381
		strong sit/layover	∞	6,800	-

Table 1: Comparison of different branching strategies for daily problems

Num. rows	Num. cols	Branching rule	Solution value	Num. nodes	Node index of the best solution
653	171836	default	7805	3300	532
		strong follow-on	7769	600	57
653	80472	default	∞	5000	-
		strong follow-on	8120	800	61

Table 2: Comparison of different branching strategies for weekly problems

Fleet name	Problem type	Number of legs	Follow-on FTC	Timeline FTC	Previous best FTC
fl1	daily	342	2.86%	2.47%	3.43%
fl2	daily	449	0.31%	0.24%	0.93%
fl3	weekly	654	6.60%	6.60%	10.0%

Table 3: FTC results

Fleet name	LP value	LP heuristic value	IP value	# major LP iterations	# fixed follow-ons	Exe. time (hrs)
fl1	637	767	1215	11	161	8
fl2	40	57	209	14	230	10
fl3	6349	6349	7805	12	0	15

Table 4: LP, IP values and computational times

The column "Follow-on FTC" refers to the FTC we obtained by using the strong follow-on branching rule in the last phase of the algorithm whereas the "Timeline FTC" column reports the FTC by using the strong sit/layover branching rule. The column "Previous best FTC" reports solutions obtained with a branch-and-price algorithm, Vance et al. [24]. Clearly our solutions substantially improve upon the existing solutions. For the $fl2$ instance our solution is 3 times better.

Table 4 reports the gaps and execution times for the algorithm based on strong follow-on branching. The column "LP heuristic value" reports the LP objective value after the column reduction heuristic was applied. For the $fl3$ problem the final integer program is obtained by taking columns with the best reduced cost (no column reduction step). In "# major LP iterations" we list the number of iterations in the LP phase. The column "# fixed follow-ons" lists the number of follow-ons that have been fixed in the column reduction heuristic. As we can see, the gaps are large; for the $fl2$ instance, the (IP-LP)/LP gap is more than 500%. We do not know whether the true gaps are smaller. Solutions can possibly be improved by using a parallel branch-and-bound solver in the final phase, however we believe that the gap would remain big.

Table 5 reports a breakdown of execution times averaged over 3 instances for the algorithm based on strong follow-on branching. The pairing generation routine is scalable as demonstrated in Klabjan and Schwan [19]. The parallel primal-dual simplex algorithm achieves good speedups on a moderate number of processors, Klabjan, Johnson and Nemhauser [18]. The last phase still has room for improvements as only the strong branching routine is carried out in parallel. The node processing is performed

Phase	Execution time	Number of processors
pairing generation	34%	96
LP solving	15%	12
column reduction	11%	12
IP solving	40%	36

Table 5: The breakdown of execution times

sequentially. Nevertheless, the overall computational time is reasonably low on a parallel architecture that has good cost/performance ratio. Also note that a substantial number of processors is used only in the pairing generation phase.

We have tried to consider a smaller number of pairings for the last stage but with no success. Either the resulting problem was infeasible or the solution produced was inferior. After taking 15,000 columns as suggested in Chu, Gelman and Johnson [11] and Anbil, Johnson and Tanga [3], the problems were infeasible after evaluating a substantial number of nodes. We believe that the difference is in the quality of the LP solution. In their problems the gap is below 2%. Such a gap for our problems would imply solutions that are on average 15 times better than those from [24] which is unlikely. For larger daily fleets we have tried either taking 100,000 low reduced cost pairings or applying the column reduction heuristic. The latter produced a better result, justifying the heuristic.

7 Concluding Remarks

Our random column selection ideas can be incorporated into a branch-and-price approach to crew scheduling. In this framework pricing is slow and randomly selecting connections might yield improvements.

We also believe that the overall methodology can be applied to other set partitioning problems like vehicle routing and cutting stock problems. The only step that has to be completely adapted is the random generation of columns. For cutting stock problems, Markov chain sampling techniques can be used to generate columns, Dyer et al. [14]. The strong follow-on branching rule can be generalized to a 'strong Ryan-Foster branching rule'.

8 Acknowledgments

This work was supported by NSF grant DMI-9700285 and United Airlines. Intel Corporation funded the parallel computing environment and ILOG provided the linear programming solver used in the computational experiments.

References

[1] Anbil, R., Forrest, J. and Pulleyblank, W. 1998. Column Generation and the Airline Crew Pairing Problem, *Extra Volume Proceedings ICM*. Available from http://www.math.uiuc.edu/documenta/xvol-icm/17/17.html.

[2] Anbil, R., Gelman, E., Patty, B. and Tanga, R. 1991. Recent Advances in Crew Pairing Optimization at American Airlines. *Interfaces*, **21**, 62–74.

[3] Anbil, R., Johnson, E. and Tanga, R. 1992. A Global Approach to Crew Pairing Optimization. *IBM Systems Journal*, **31**, 71–78.

[4] Andersson, E., Housos, E., Kohl, N. and Wedelin, D. 1998. Crew Pairing Optimization. In *Operations Research in the Airline Industry*. G. Yu (editor). Kluwer Academic Publishers, 228–258.

[5] Barnhart, C., Johnson, E., Nemhauser, G., Savelsbergh, M. and Vance, P. 1998. Branch-and-Price: Column Generation for Solving Huge Integer Programs. *Operations Research*, **46**, 316–329.

[6] Barnhart, C., Johnson, E., Nemhauser, G. and Vance, P. 1999. Crew Scheduling. In *Handbook of Transportation Science*. R. W. Hall (editor). Kluwer Scientific Publishers, 493–521.

[7] Beale, E. and Tomlin, J. 1970. Special Facilities in a General Mathematical Programming System for Non-Convex Problems Using Ordered Sets of Variables, *Proceedings of the 5th International Conference on Operations Research*.

[8] Bertsekas, D. 1995. *Nonlinear Programming*, Athena Scientific, 79–90.

[9] Bixby, R., Cook, W., Cox, A. and Lee, E. 1995. Parallel Mixed Integer Programming, *Technical Report CRPC-TR95554*, Rice University. Available from ftp://softlib.rice.edu/pub/CRPC-TRs/reports.

[10] Bixby, R., Gregory, J., Lustig, I., Marsten, R. and Shanno, D. 1992. Very Large-scale Linear Programming: A Case Study in Combining Interior Point and Simplex Methods. *Operations Research*, **40**, 885–897.

[11] Chu, H., Gelman, E. and Johnson, E. 1997. Solving Large Scale Crew Scheduling Problems. *European Journal of Operational Research*, **97**, 260–268.

[12] CPLEX Optimization 1997. *Using the CPLEX Callable Library*, 5.0 edn, ILOG Inc.

[13] Desrosiers, J., Dumas, Y., Desrochers, M., Soumis, F., Sanso, B. and Trudeau, P. 1991. A Breakthrough in Airline Crew Scheduling, *Technical Report G-91-11*, Cahiers du GERAD.

[14] Dyer, M., Frieze, A., Kapoor, A., Kannan, R., Perkovic, L. and Vazirani, U. 1994. A Mildly Exponential Time Algorithm for Approximating the Number of Solutions to a Multidimensional Knapsack Problem. Unpublished.

[15] Feo, T. and Resende, M. 1989. A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. *Operations Research Letters*, **8**, 67–71.

[16] Gershkoff, I. 1989. Optimizing Flight Crew Schedules. *Interfaces*, **19**, 29–43.

[17] Klabjan, D. 1999. *Topics in Airline Crew Scheduling and Large Scale Optimization*. Ph.D. Dissertation, Georgia Institute of Technology.

[18] Klabjan, D., Johnson, E. and Nemhauser, G. 1999. A Parallel Primal-Dual Algorithm, *Technical Report TLI/LEC-99-10*, Georgia Institute of Technology. To appear in *Operations Research Letters*.

[19] Klabjan, D. and Schwan, K. 1999. Airline Crew Pairing Generation in Parallel, *Technical Report TLI/LEC-99-02*, Georgia Institute of Technology.

[20] Law, A. and Kelton, W. 1991. *Simulation, Modeling and Analysis*. McGraw-Hill.

[21] Linderoth, J. and Savelsbergh, M. 1999. A Computational Study of Search Strategies for Mixed Integer Programming. *Informs Journal on Computing*, **11**, 173–187.

[22] Message Passing Interface Forum 1995. *The MPI Message Passing Standard*. Available from http://www.mpi-forum.org.

[23] Ryan, D. and Foster, B. 1981. An Integer Programming Approach to Scheduling. In *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*. A. Wren (editor). North-Holland, 269–280.

[24] Vance, P., Atamtürk, A., Barnhart, C., Gelman, E., Johnson, E., Krishna, A., Mahidhara, D., Nemhauser, G. and Rebello, R. 1997. A Heuristic Branch-and-Price Approach for the Airline Crew Pairing Problem, *Technical Report LEC-97-06*, Georgia Institute of Technology.