[1]A New Pricing Scheme for Airline Crew Scheduling

Alexandra Makri (makri@uiuc.edu)
Diego Klabjan (klabjan@uiuc.edu)
Department of Mechanical and Industrial Engineering
University of Illinois at Urbana-Champaign
Urbana, IL

Abstract

Solving LP relaxations of airline crew scheduling models is computationally challenging due to a large number of variables, complex feasibility rules to generate columns, and nonlinear cost. We perform computational experiments with a nonlinear pricing strategy. We develop a column generation scheme that uses several pruning rules to fathom column enumeration. The pruning rules are categorized into approximate and exact, where the approximate rules might prune columns that would yield an improved objective value. The pruning rules use the fact that columns are paths in a network and we use shortest path algorithms and their extensions to obtain bounds.

1 Introduction

The *airline crew scheduling problem* is the problem of finding crew itineraries or *pairings* that minimize the crew cost. Traditionally the crew scheduling problem is modeled as the set partitioning problem $\min\{cx : Ax = \mathbf{1}, x \text{ binary}\}$, where each variable corresponds to a pairing and each row to a leg, a_{ij} = 1 if leg i is in pairing j and 0 otherwise, and c_j is the cost of pairing j. This model is computationally difficult due to the large number of pairings, their complex structure and the nonlinear cost function. Medium size U.S. domestic problems of major carriers with approximately 300 legs have billions of variables. These facts make even solving LP relaxations $\min\{cx : Ax = \mathbf{1}, x \geq 0\}$ a strenuous task. In this paper we consider only the LP relaxation. (See Barnhart et. al. (1999) for a survey on how to obtain a binary solution.)

Many algorithms for airline crew scheduling require solving the LP relaxation many times, e.g. branch-and-price. Due to the large number of variables, i.e. pairings, explicitly solving the LP relaxation is usually intractable. Instead column generation is used, where in every iteration a restricted master problem, called also the *subproblem*, is solved over a small number of pairings. The subproblem is the same problem as the original problem except that it consists of only a subset of the columns. Pairings with low reduced cost are then appended to the subproblem, the subproblem is reoptimized, and the procedure is repeated. A formal description of the column generation algorithm follows.

- Loop
  - Solve the LP over a small subset S of the columns, i.e. solve the subproblem over S.
  - Find a set $\bar{S}$ of columns with low reduced cost. This step is called *pricing*.

---

[1] Partially supported by a grant from the University of Illinois at Urbana-Champaign Campus Research Board.

- If there are no columns with negative reduced cost, exit. We have an optimal solution.
- $S = S \cup \overline{S}$

- End loop

Pairings are paths in a network, where flights correspond to nodes and arcs to crew connections. A pairing is subject to various feasibility rules such as company, union and regulatory restrictions and therefore a path in the network is not necessarily a pairing. These rules constrain the paths that correspond to pairings. When solving the LP relaxations by column generation, pricing has to be carried out in every iteration. One approach to finding the lowest reduced cost pairing is by applying a constrained shortest path algorithm, Desrochers and Soumis (1988), Desaulniers et al. (1998) and Desrosiers et al. (1995). These algorithms are generalizations of shortest path algorithms. A constrained shortest path algorithm keeps track of several labels that correspond to the legality rules and the cost structure (called also resources) and they are updated upon scanning an arc. On the other hand, pairing enumeration methods attempt to generate the pairings in a depth-first search manner. Typically some ad-hoc rules are used to reduce the computation time. Anderson et al. (1998) describe a commercial crew scheduling software that uses enumeration. In the presence of many labels a constrained shortest path algorithm is likely to enumerate all the pairings and therefore it behaves like an enumeration algorithm. The main difference between the two approaches is that in a constrained shortest path algorithm the nodes are scanned more in a breath-first search order and therefore it has larger storage requirements. We chose to perform pricing based on enumeration but the approach presented applies to a constrained shortest path pricing as well (see Section 4).

Bixby et al. (1992) develop a new strategy to append columns to the subproblem. Instead of adding columns with low reduced cost they propose to add columns with a low ratio of cost over sum of the dual prices in the column. We call this ratio the score. They use this strategy on problems, where all of the columns are explicitly given in advance and they obtain a reduced number of pricing calls. We embed this rule in a column generation algorithm, where in pricing columns are enumerated by depth-first search. We design several pruning rules that fathom enumeration and reduce the generation time. The approximate pruning rules are applied in initial iterations and they prune pairings that might improve the objective value. After the approximate rules do not improve the objective value anymore, we switch to exact pruning rules. These rules prune the generation by providing an upper bound on the score. The main contributions of this work are the embedding of a nonlinear pricing rule in column generation, the idea of pruning too many pairings in early iterations to expedite the generation, and the development of several pruning rules that speed up the generation procedure.

Next we list some relevant publications on airline crew scheduling; for an extended list see e.g. Barnhart et al. (1999). Desrosiers et al. (1991) solve the crew scheduling problem using column generation and a constrained shortest path algorithm for pricing. This is the first work that uses constrained shortest path for airline crew scheduling. Early work in airline crew scheduling is the so-called TRIP algorithm, Gershkoff (1989), Anbil et al. (1991). TRIP is a local search heuristic that uses pairing enumeration on a subset of flights. Anbil, Johnson and Tanga (1991) use column

generation to solve large-scale airline crew scheduling problems. In pricing they enumerate pairings. Bixby et al. (1992) propose a new pricing rule and they experiment with different LP algorithms for solving the subproblem. Vance et al. (1997) and Anbil, Forrest and Pulleyblank (1998) describe column generation, branching and search strategies for a branch-and-price algorithm. The former publication uses a constrained shortest path algorithm in pricing and the latter uses enumeration in pricing. Klabjan, Johnson and Nemhauser (2001) use generation in pricing. Their generation is based on randomness, i.e. they generate random pairings.

Section 2 presents the new pricing rule and all the pruning rules. In Section 3 we report the computational experiments. We conclude the introduction with a detailed description of airline crew scheduling and we describe the underlying network.

Airline Crew Scheduling

The input for an airline crew scheduling problem is a fleet together with the flight schedule and the aircraft routes. A *leg* is a nonstop flight.

A *duty* is a working day of a crew and it consists of a sequence of legs. A connection within a duty is called a *sit connection*. There is a lower and an upper limit on the sit connection time. The minimum sit connection time can be violated only if the crew stays on the same plane. Company, union and regulatory restrictions restrict a duty. The cost of a duty, measured in minutes, is usually the maximum of three quantities: the flying time, a fraction of the elapsed time, and the duty minimum guarantee pay. We denote by dc_d the cost of duty d.

Crew bases are designated stations where crews are based. A *pairing* is a sequence of duties, starting and ending at the same crew base. A connection within a pairing is called an *overnight connection* or *layover*. Like sit connections there is a lower and an upper limit on the layover time, denoted by *minRest* and *maxRest* respectively. A pairing is also subject to many feasibility rules, e.g. maximum elapsed time, maximum number of duties, service of particular legs from a given crew base. We denote by M the maximum number of duties in a pairing. Typical values of M are between 3 and 5. The cost of a pairing p, denoted by pc_p and measured in minutes, is the maximum of three quantities: the sum of the duty costs in the pairing, a fraction f times the elapsed time, and a minimum guaranteed pay mg, which can be different from the duty minimum guarantee pay, times the number of duties. Formally,

$$pc_p = \max\{mg \cdot nd, f \cdot pel, \sum_{d \in p} dc_d\}, \qquad (1)$$

where nd is the number of duties in p and pel is the elapsed time, called also time-away-from-base. A detailed discussion of the legality rules and the cost structure of pairings is given by Barnhart et al. (1999).

The *daily crew scheduling problem* is the crew scheduling problem with the assumption that every leg is flown every day of the week. In the *weekly crew scheduling problem* a crew schedule is sought over a time horizon that is repeated, i.e. it is assumed that the flight schedule repeats. Pairings from the end of the horizon can wrap around to the beginning of the horizon. The *dated crew scheduling problem* is similar to the weekly problem except that the flight schedule is given over a fixed time horizon. In a dated problem pairings do not wrap around. For a discussion of the applications of these three problems see e.g. Barnhart et al. (1999). The methodologies developed in this paper are applied to all three problems.

The Mixed Segment/Duty Timeline Network

The *mixed segment/duty timeline network* has two nodes for each leg, one corresponding to the arrival and the other one corresponding to the departure of the leg. We denote by arr_i, dep_i the arrival, departure node corresponding to leg i, respectively. The network has two types of arcs. For each duty d there is a *duty arc* that connects dep_i with arr_j, where i is the first leg in d and j is the last leg in d. In addition, there is a *connection arc* (arr_i, dep_j) if the arrival station of leg i is the same as the departure station of leg j and the connection time is within $[minRest, maxRest]$. The connection time is defined as the departure time of leg j minus the arrival time of leg i. Note that this network is acyclic and it has parallel duty arcs. Pairings are paths in the mixed segment/duty timeline network, but due to the feasibility rules a path does not necessarily correspond to a pairing. This network is a mixture between the traditional segment timeline network and the duty timeline network, see e.g. Barnhart et al. (1999) for a description of these two networks. It can be stored much more compactly than the duty timeline network since the nodes correspond to the nodes of the segment timeline network but on the other hand it does not capture as many feasibility rules as the duty timeline network.

Example. Consider the mixed segment/duty timeline network shown in Figure 1.

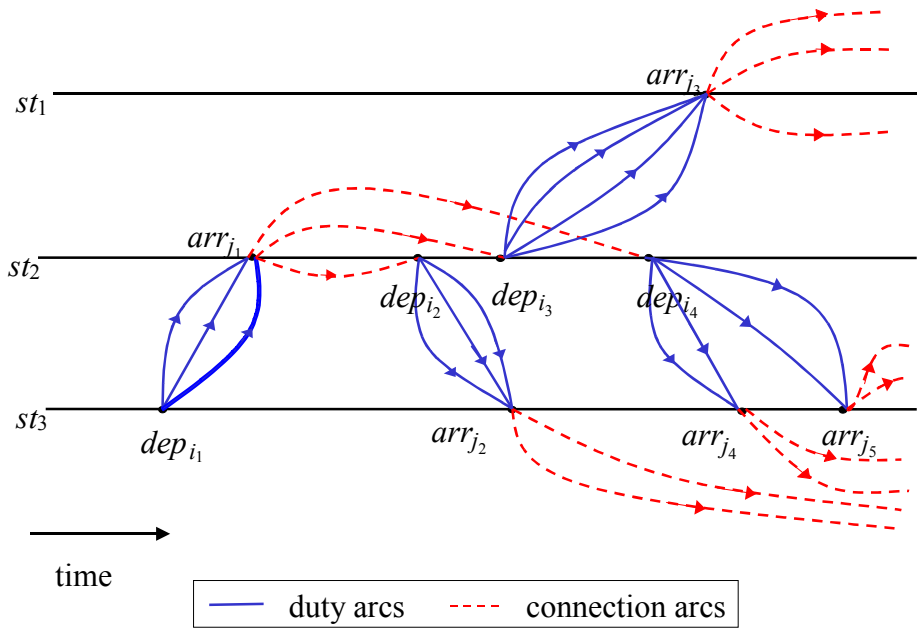

Figure 1. A mixed segment/duty timeline network

We assume that the departure station of leg i_1 is st_3, the arrival station of leg j_1 is st_2, etc. The solid arcs correspond to the duty arcs and the dashed arcs correspond to the connection arcs. For example, there are 3 duties with leg i_1 as the first leg in the duty and leg j_1 as the last leg in the duty. The time between the departure of leg i_2 and the arrival of leg j_1 is greater or equal to *minRest*. Similarly, the time between the departure of leg i_4 and the arrival of leg j_1 is less or equal to *maxRest*.

2 Pricing and pruning

2.1 Fractional pricing

In the column generation algorithm, in every iteration columns are added to the subproblem. Let y be an optimal dual vector to the subproblem, i.e. y is the optimal dual vector to $\min\{\bar{c}x : \bar{A}x = \mathbf{1}, x \geq 0\}$, where $\bar{A}$ is a submatrix of A and $\bar{c}$ the corresponding coordinates in c. In traditional pricing, pairings p with the smallest reduced cost $c_p - \sum_{i \in p} y_i$ are added to the subproblem since the reduced cost approximates the change in the objective value if the column is appended to the subproblem. Bixby et al. (1992) observed that the number of pricing calls decreases if columns p with low *score* $s_p = c_p \Big/ \sum_{i \in p} y_i$ are selected. The main objective of this work is to show how to find columns with low score within the column generation framework. Note that since the pairing cost is nonnegative, a pairing p has negative reduced cost if and only if $\sum_{i \in p} y_i > 0$ and $s_p < 1$. Therefore in pricing the goal is to solve

$$\min_p \left\{ s_p \mid \sum_{i \in p} y_i > 0 \right\}. \tag{2}$$

If this minimum is greater or equal to 1, then the current solution is optimal.

Note that $c_p \Big/ \sum_{i \in p} y_i = \left(c_p - \sum_{i \in p} y_i \right) \Big/ \sum_{i \in p} y_i + 1$. The intuition why score based pricing is more efficient is that the columns with low score have low reduced cost and in the presence of negative reduced cost they tend to have small $\sum_{i \in p} y_i$. By LP sensitivity analysis we clearly want the former property. Pairings obeying the latter property tend to have a smaller number of legs and therefore these 'short' pairings are likely to blend better with other pairings. Therefore score pricing distinguishes among the pairings with the same negative reduced cost by giving preference to those with small $\sum_{i \in p} y_i$ and therefore smaller number of legs. Note that pairings with small $\sum_{i \in p} y_i$ tend to have a small number of legs.

2.2 Pruning

To enumerate all of the pairings in every iteration of a column generation algorithm is too time consuming due to the large number of pairings and complex feasibility rules. A *partial pairing* is a sequence of duties that starts at a crew base and meets all the pairing feasibility rules except that it does not necessarily end at the same station. The pairings are generated using depth-first search on the mixed segment/duty timeline network. In the depth-first search procedure a partial pairing is extended with all possible duties and the procedure is recursively repeated. When a pairing is found, i.e. the partial pairing starts and ends at the same crew base, the pairing is stored and we backtrack by removing the last duty from the pairing. If we knew in advance that a partial pairing leads only to pairings with score greater or equal to 1, then we would stop depth-first search from the

current partial pairing and backtrack. *Pruning* is a procedure that fathoms depth-first search of a partial pairing before a pairing is actually obtained. For example, given a partial pairing, if we somehow know that all the pairings p resulting from this partial pairing will have $\sum_{i \in p} y_i$ smaller or equal to 0, we can abort the generation of this partial pairing and backtrack. Cleary efficient pruning rules can substantially reduce the execution time of the generation procedure.

We develop two types of pruning rules, which we call approximate and exact. Approximate pruning rules prune partial pairings that can produce pairings with score greater or equal to 1. These rules prune many partial pairings and therefore produce fast generation. However, they might prune partial pairings that would yield pairings with score less than 1. If after using the approximate pruning rules we do not find pairings with score less than 1, we cannot state that the incumbent primal solution is optimal. On the other hand, exact pruning rules prune only partial pairings that would always result in pairings with score greater or equal to 1 (in the worst case scenario). We use first the approximate pruning rules and then we switch to the exact pruning rules, when we do not find a pairing with score less than 1 by using approximate pruning.

The idea behind this distinction is that it really does not matter how we prune in all of the iterations but the last one. In the last iteration, where we actually prove optimality, we cannot prune partial pairings approximately. A typical implementation along these lines would have a constant K and it would prune all the partial pairings with current score greater or equal to K. It is hard to make such a strategy flexible for all possible instances, feasibility rules and cost structures, i.e. K should depend on these parameters. Our approximate rules are designed in such a way that they use the properties of the mixed segment/duty timeline network.

2.2.1 Approximate pruning rules

Let p be a partial pairing and let $|p|$ be the number of duties in the partial pairing. For a duty d we denote by $dy_d = \sum_{i \in d} y_i$ the sum of the dual prices of the legs in the duty and we call it the *dual value of duty d*. Note that p represents the set of all the duties in the pairing and likewise d stands for the set of all the legs in the duty. Based on the quality of the approximation we have two stages. In the approximate stage 1 we prune many pairings and in the approximate stage 2, which follows the approximate stage 1, we refine pruning. Next we list the approximate pruning rules.

Approximate stage 1

Let $\overline{dy} = \sum_d dy_d \Big/ n$ and $\overline{dc} = \sum_d dc_d \Big/ n$ be the average of the duty dual values and the duty cost over all the duties, respectively. Here n is the number of the duties. The pruning rules in the approximate stage 1 use the average sum of the duty dual values and the average duty cost to prune partial pairings. Therefore the approximation is rather coarse.

- Approximate pruning rule 1: If for a partial pairing p we have $\sum_{d \in p} dy_d + (M - |p|)\overline{dy} \le 0$, then we prune p.

Here we make the assumption that the duty dual value of every duty equals to the average $\overline{dy}$. If this assumption holds and $\sum_{d\in p} dy_d + (M - |p|)\overline{dy} \le 0$, then for every pairing $\overline{p}$ resulting from p we have $\sum_{i\in \overline{p}} y_i \le 0$ and $\overline{p}$ would not be considered in (2).

- Approximate pruning rule 2: If for a partial pairing p we have

$$\left(\sum_{d\in p} dc_d + (M - |p|)\overline{dc}\right) \Big/ \left(\sum_{d\in p} dy_d + (M - |p|)\overline{dy}\right) \ge 1,$$

then we prune p.

The approximations here are first that the pairing cost is the sum of the duty costs in the pairing, second that the duty cost for every duty equals to the average duty cost, and third that the duty dual value of every duty equals to the average duty dual value. Under these assumptions the score of every pairing resulting from p would be greater or equal to 1. By applying this rule after the approximate pruning rule 1, the denominator is always greater than 0.

Approximate stage 2

We say that a path r is a *path from leg i to crew base cb* if r is a path in the mixed segment/duty timeline network from arr_i to a node in $\{arr_k \mid$ the arrival station of leg k is at crew base $cb\}$. For a leg i, crew base cb, and j, $1 \le j \le M-1$, let $R_{i,j}^{cb}$ be the set of all paths from i to crew base cb with exactly j duty arcs. Given a path $r \in R_{i,j}^{cb}$, let cy_r be the sum of the duty dual values of all the duty arcs in r. Similarly we define cc_r with respect to the duty cost. For $r \in R_{i,j}^{cb}$, let cel_r be the sum of the connection times of all the connection arcs in r and the duty elapsed times of all the duty arcs in r. By appropriately defining the arc cost in the mixed segment/duty timeline network, these values correspond to the cost of a path in the network.

Example. (cont.) Let us define the cost on solid arcs as the duty dual values and the cost on the dashed connection arcs as 0. Then cy_r is the cost of path r. Similarly we can treat cc_r. For cel_r, we define the cost of solid duty arcs as the duty elapsed time and the cost of the dashed connection arcs as the connection time. Then cel_r is the cost of path r.

For a leg i, crew base cb and for every $j, 1 \le j \le M-1$, we denote by $\overline{dy}_{i,j}^{cb}$, $\overline{dc}_{i,j}^{cb}$, and $\overline{del}_{i,j}^{cb}$ the average of cy_r, cc_r and cel_r over all paths $r \in R_{i,j}^{cb}$, respectively. In addition, let $\overline{ay}_{i,j}^{cb}$ be the average of cy_r over all paths $r \in \bigcup_{k=1}^{j} R_{i,k}^{cb}$, i.e. we restrict the paths to have at most j duty arcs. We denote by pel_p the elapsed time of a partial pairing p and let $l(p)$ be the last leg in the last duty of p. Pruning in the approximate stage 2 is less coarse than pruning from the approximate phase 1, because these average values are better approximations.

- Approximate pruning rule 3: If for a partial pairing p that starts at a crew base cb, $\sum_{d\in p} dy_d + \overline{ay}_{l(p),M-|p|}^{cb} \le 0$, then we prune p.

This rule is similar to the first approximate rule but the approximation is better since we are using the average sum of the duty dual values among the paths that connect the last leg in the last duty of the partial pairing to the crew base.

- Approximate pruning rule 4: Let p be a partial pairing and let

$$S_p = \left\{ j : 1 \le j \le M - |p|, \sum_{d \in p} dy_d + \overline{dy}^{cb}_{l(p),j} > 0 \right\},$$

where p starts at a crew base cb. If

$$\sum_{j \in S_p} \frac{\max\{mg \cdot (|p|+j), f \cdot (pel_p + \overline{del}^{cb}_{l(p),j}), \sum_{d \in p} dc_d + \overline{dc}^{cb}_{l(p),j}\}}{\sum_{d \in p} dy_d + \overline{dy}^{cb}_{l(p),j}} \Big/ |S_p| \ge 1,$$

then we prune p.

Given a fixed $j, 1 \le j \le M - |p|$, the numerator in the summand estimates the cost of pairings resulting from p by appending exactly j duties. Recall the pairing cost structure given by (1). The first term in the maximum corresponds to the minimum guarantee pay, the second term to the average elapsed time multiplied by f, and the third term to the average sum of the duty costs. The denominator in the summand is the average sum of the duty dual values if p is extended by exactly j duties. Therefore the summand approximates the average score of parings that result from p by appending exactly j duties. The overall approximation is then the average of all the approximations to the average score.

Note that if p has not been pruned by the approximate pruning rule 3, then by conditioning it is easy to see that $S_p \neq \emptyset$.

2.2.2 Computing the averages

In this section we show how to compute $\overline{dy}^{cb}_{i,j}$, $\overline{dc}^{cb}_{i,j}$ and $\overline{del}^{cb}_{i,j}$. Let $D = (N,A)$ be an acyclic network, which is already topologically sorted. To capture the mixed segment/duty timeline network we allow parallel arcs. We assume that every node i has a weight u_i and every arc e has a weight v_e. We show how to compute $\overline{a}_{i,j}$ for every $i \in N, j = 1,...,K$, where $\overline{a}_{i,j}$ is the average length of paths from node i to a given node t among all the paths with precisely j arcs. K is a given constant. The length of a path is defined as the sum of the node and the arc weights in the path.

Let $R_{i,j}$ be the set of all paths from node i to node t with exactly j arcs and for simplicity of notation we denote $n_{i,j} = |R_{i,j}|$. We define $\overline{a}_{i,0} = n_{i,0} = 0$ for every $i \in N \setminus \{t\}$ and $n_{t,0} = 1, \overline{a}_{t,0} = 0, \overline{a}_{t,j} = n_{t,j} = 0$ for j=1,…,K. Given $i \in N$, the notation $\sum_{(i,l) \in A}$ represents the sum over all arcs originating at i. Note that due to the parallel arcs this summation might be different from the summation over all neighbors l. Then by definition of average for every $i \in N \setminus \{t\}$ and for every j=1,…,K we have

$$\overline{a}_{i,j} = \frac{\sum_{r\in R_{i,j}} (\sum_{g\in r} u_g + \sum_{k\in r} v_k)}{n_{i,j}} = \frac{\sum_{(i,l)\in A}\sum_{r\in R_{l,j-1}} (u_i + v_{(i,l)} + \sum_{g\in r} u_g + \sum_{k\in r} v_k)}{n_{i,j}}$$

$$= \frac{u_i \sum_{(i,l)\in A}\sum_{r\in R_{l,j-1}} 1 + \sum_{(i,l)\in A}\sum_{r\in R_{l,j-1}} (v_{(i,l)} + \sum_{g\in r} u_g + \sum_{k\in r} v_k)}{n_{i,j}}$$

$$= u_i + \frac{\sum_{(i,l)\in A} v_{(i,l)} \sum_{r\in R_{l,j-1}} 1 + \sum_{(i,l)\in A}\sum_{r\in R_{l,j-1}} (\sum_{g\in r} u_g + \sum_{k\in r} v_k)}{n_{i,j}}$$

$$= u_i + \frac{\sum_{(i,l)\in A} v_{(i,l)} n_{l,j-1} + \sum_{(i,l)\in A} \overline{a}_{l,j-1} n_{l,j-1}}{n_{i,j}} = u_i + \frac{\sum_{(i,l)\in A} (v_{(i,l)} + \overline{a}_{l,j-1}) n_{l,j-1}}{n_{i,j}}.$$

Clearly $n_{i,j} = \sum_{(i,l)\in A}\sum_{r\in R_{l,j-1}} 1 = \sum_{(i,l)\in A} n_{l,j-1}$.

Based on these formulas we develop the algorithm, which is given in Figure 2. Without loss of generality we assume that t is the last node in the topological order. Since D is topologically sorted, it is easy to verify the correctness of the algorithm. We compute $\overline{dy}^{cb}_{i,j}$, $\overline{dc}^{cb}_{i,j}$ and $\overline{del}^{cb}_{i,j}$ by running the algorithm several times on the mixed segment/duty timeline network and with different arc costs. We run it once for each crew base. For each run the input mixed segment/duty timeline network is slightly modified by removing all the connection arcs with the tail starting at the crew base *cb*. In addition, we add an artificial node and we connect each node arr_i with the arrival station of leg i equal to *cb* to this artificial node.

$\overline{ay}^{cb}_{i,j}$ is computed from $\overline{dy}^{cb}_{i,j}$ and $n^{cb}_{l,j}$ by using the conditioning formula

$$\overline{ay}^{cb}_{i,j} = \sum_{l=1}^{j} \overline{dy}^{cb}_{i,l} n_{i,l} \Big/ \sum_{k=1}^{K} n_{i,k}.$$

Input: $D=(N,A)$
Output: $\overline{a}_{i,j}$ for all $i\in N, j=1,\ldots,K$.

$n = \overline{a} = 0, n_{t,0} = 1$
for $i=|N|,\ldots,1$ **do**
  for $j=1,\ldots,K$ **do**
    for all $(k,i)\in A$ **do**
      $n_{k,j} = n_{k,j} + n_{i,j-1}$
    end for
  end for
  $f = 0$
  for $j=1,\ldots,K$ **do**
    for all $(i,k)\in A$ **do**
      $f = f + n_{k,j-1}(\overline{a}_{k,j-1} + v_{(i,k)})$

end for

$$\overline{a}_{i,j} = u_i + \frac{f}{n_{i,j}}$$

end for

end for

Figure 2: The algorithm for computing the averages

2.2.3 Exact pruning rules

Assume that in an iteration of the column generation algorithm we did not find a pairing with the score less than 1 by applying approximate pruning rules. Since these rules might have pruned too much, we cannot assert that the current solution is optimal. Therefore we need to develop pruning rules that provide upper bounds on the score.

We denote the maximum allowed flying time in a duty by F and by dfl_d the flying time of a duty d. For a leg i, crew base cb, and $j, 1 \le j \le M-1$, let $u^{cb}_{i,j}$ be the length of the longest path, with respect to the sum of the duty dual values along the path, among all paths $r \in \bigcup_{k=1}^{j} R^{cb}_{i,k}$. In addition, for a leg i, crew base cb, and $j, 1 \le j \le M-1$, we define

$$\tau^{cb}_{i,j} = \max\left\{ \max\left\{ \frac{\sum_{k \in \tilde{d}(r)} dy_k}{\sum_{k \in \tilde{d}(r)} dfl_k} \;\middle|\; r \in \bigcup_{k=1}^{j} R^{cb}_{i,k} \right\}, 0 \right\},$$

where $\tilde{d}(r)$ is the set of all the duty arcs in path r. Next we give the pruning rules.

- Exact pruning rule 1: If for a partial pairing p that starts at a crew base cb we have $\sum_{d \in p} dy_d + u^{cb}_{l(p),M-|p|} \le 0$, then we can prune the partial pairing.

Recall that a pairing is a path in the mixed segment/duty timeline network. If $\overline{p}$ is a pairing resulting from p, then by the definition of u we have

$$\sum_{d \in \overline{p}} dy_d = \sum_{d \in p} dy_d + \sum_{d \in \overline{p} \setminus p} dy_d \le \sum_{d \in p} dy_d + u^{cb}_{l(p),M-|p|} \le 0$$

and therefore $\overline{p}$ would not be considered in (2).

- Exact pruning rule 2: If for a partial pairing p that starts at a crew base cb we have $\sum_{d \in p} dy_d \ge 0$ and

$$\left(\frac{\sum_{d \in p} dc_d + (M - |p|)F}{\sum_{d \in p} dfl_d + (M - |p|)F} \right) \Bigg/ \left(\frac{\sum_{d \in p} dc_d}{\sum_{d \in p} dfl_d} + \tau^{cb}_{l(p),M-|p|} \right) \ge 1 ,$$

then we prune p.

In order to prove the validity of this pruning rule, we have to show that any pairing resulting from p has a score greater or equal to 1. Let $\overline{p}$ be such a pairing. Then

$$s_{\bar{p}} = \frac{pc_{\bar{p}}}{\sum_{d\in\bar{p}} dy_d} = \frac{pc_{\bar{p}}}{\sum_{d\in\bar{p}} dfl_d} \Big/ \frac{\sum_{d\in\bar{p}} dy_d}{\sum_{d\in\bar{p}} dfl_d} \geq \frac{\sum_{d\in p} dc_d + (M-|p|)F}{\sum_{d\in p} dfl_d + (M-|p|)F} \Big/ \frac{\sum_{d\in\bar{p}} dy_d}{\sum_{d\in\bar{p}} dfl_d}, \quad (3)$$

where the inequality is proven in Klabjan, Johnson, and Nehmhauser (2001). Let $\alpha = \sum_{d\in p} dy_d$, $\beta = \sum_{d\in\bar{p}\setminus p} dy_d$, $\omega = \sum_{d\in p} dfl_d$ and $\sigma = \sum_{d\in\bar{p}\setminus p} dfl_d$.

If $\beta < 0$, then

$$\frac{\sum_{d\in\bar{p}} dy_d}{\sum_{d\in\bar{p}} dfl_d} = \frac{\alpha+\beta}{\omega+\sigma} \leq \frac{\alpha}{\omega} + \tau^{cb}_{l(p),M-|p|}$$

since from $\beta < 0$, $\tau^{cb}_{l(p),M-|p|} \geq 0$, and $\alpha \geq 0$ it follows $\beta\omega \leq 0 \leq \alpha\sigma + \tau^{cb}_{l(p),M-|p|}\omega^2 + \tau^{cb}_{l(p),M-|p|}\omega\sigma$. If $\beta \geq 0$, then

$$\frac{\sum_{d\in\bar{p}} dy_d}{\sum_{d\in\bar{p}} dfl_d} = \frac{\alpha+\beta}{\omega+\sigma} \leq \frac{\alpha}{\omega} + \frac{\beta}{\sigma} \leq \frac{\alpha}{\omega} + \tau^{cb}_{l(p),M-|p|},$$

where the first inequality follows from $\alpha \geq 0$, $\beta \geq 0$ and the second one from the definition of τ and the fact that $\bar{p}\setminus p$ corresponds to a path in the mixed segment/duty timeline network. Together we get that

$$\frac{\sum_{d\in\bar{p}} dy_d}{\sum_{d\in\bar{p}} dfl_d} \leq \frac{\sum_{d\in p} dy_d}{\sum_{d\in p} dfl_d} + \tau^{cb}_{l(p),M-|p|}. \quad (4)$$

From (3) and (4) we obtain

$$s_{\bar{p}} \geq \frac{\sum_{d\in p} dc_d + (M-|p|)F}{\sum_{d\in p} dfl_d + (M-|p|)F} \Big/ \frac{\sum_{d\in\bar{p}} dy_d}{\sum_{d\in\bar{p}} dfl_d} \geq \left(\frac{\sum_{d\in p} dc_d + (M-|p|)F}{\sum_{d\in p} dfl_d + (M-|p|)F}\right) \Big/ \left(\frac{\sum_{d\in p} dy_d}{\sum_{d\in p} dfl_d} + \tau^{cb}_{l(p),M-|p|}\right) \geq 1.$$

Therefore the score of $\bar{p}$ is greater or equal to 1.

- Exact pruning rule 3: If for a partial pairing p that starts at crew base *cb*, we have $\sum_{d\in p} dy_d < 0$ and $\frac{\sum_{d\in p} dc_d + (M-|p|)F}{\sum_{d\in p} dfl_d + (M-|p|)F} \Big/ \tau^{cb}_{l(p),M-|p|} \geq 1$, then we prune p.

Let $\bar{p}$ be a partial pairing resulting from p with $\sum_{d\in\bar{p}} dy_d > 0$. Since $\sum_{d\in p} dy_d < 0$ it follows that $\sum_{d\in\bar{p}\setminus p} dy_d > 0$. Therefore by the definition of τ we have

$$\frac{\sum_{d\in\bar{p}} dy_d}{\sum_{d\in\bar{p}} dfl_d} = \frac{\sum_{d\in p} dy_d + \sum_{d\in\bar{p}\setminus p} dy_d}{\sum_{d\in p} dfl_d + \sum_{d\in\bar{p}\setminus p} dfl_d} \leq \frac{\sum_{d\in\bar{p}\setminus p} dy_d}{\sum_{d\in\bar{p}\setminus p} dfl_d} \leq \tau^{cb}_{l(p),M-|p|}. \quad (5)$$

From (3) and (5) we obtain

$$s_{\bar{p}} = \frac{pc_{\bar{p}}}{\sum_{d\in\bar{p}} dy_d} \geq \frac{\sum_{d\in p} dc_d + (M-|p|)F}{\sum_{d\in p} dfl_d + (M-|p|)F} \Bigg/ \frac{\sum_{d\in\bar{p}} dy_d}{\sum_{d\in\bar{p}} dfl_d} \geq \frac{\sum_{d\in p} dc_d + (M-|p|)F}{\sum_{d\in p} dfl_d + (M-|p|)F} \Bigg/ \tau^{cb}_{l(p),M-|p|} \geq 1.$$

Therefore the score of $\bar{p}$ would be greater or equal to 1, and p can be pruned.

If p has not been pruned by the exact pruning rule 1, then since $\sum_{d\in p} dy_d < 0$ it follows that $u^{cb}_{l(p),M-|p|} > 0$ and therefore by definition $\tau^{cb}_{l(p),M-|p|} > 0$.

2.3 Pricing Methodology

Here we summarize how the pruning rules are carried out. We start by using the approximate rules 1 and 2 in pricing. Given a partial pairing we first check the approximate rule 1 and then the approximate rule 2. Every pairing with the score less than 1 is stored in a pool. If at the end of a single pairing generation pass the pool is non empty, we add a subset of columns with the score less than 1 to the subproblem (see Section 3.1 how this subset is selected) and we keep using the same pruning rules in the iteration that follows. If the pool is empty, i.e. we did not find a pairing with score less than 1, then we switch to using the approximate rules 3 and 4. Given a partial pairing, we first apply the approximate rule 3 and then the approximate rule 4. We store pairings with score less than 1 in the pool. If the pool is non empty, then we add a subset of the pairings with the score less than 1 to the subproblem and in the next iteration we keep using the same two pruning rules. If the pool is empty, then we start using the exact pruning rules. Given a partial pairing we first apply the exact pruning rule 1, followed by the exact pruning rule 2 and at the end we apply the exact pruning rule 3. The exact pruning rules are applied until an optimal solution is found. Note that in order to avoid divisions by 0, we have to apply the pruning rules in this specific order.

3 Computational Experiments

3.1 Implementation

Our pairing generation is based on the mixed segment/duty timeline network. For daily problems the network consists of several copies of duties, which are shifted by days. We add enough copies that we are able to generate every pairing. Similarly, for weekly problem, we add copies of duties at the end of the time horizon to enable the generation of pairings that wrap around.

The τ values that are needed for the exact pruning rules are computed by using an algorithm from Makri and Klabjan (2001). This reference contains three algorithms for computing τ. For sparse problems, which reflect our mixed segment/duty timeline network, the most efficient algorithm is the parametric longest path algorithm. In the definition of τ the paths are restricted to at most a given number of arcs in the paths. The algorithms in Makri and Klabjan (2001) do not handle path restrictions to a given number of arcs and due to the nonlinearity in the objective function the traditional dynamic programming approach cannot be applied. To circumvent this, we explicitly restrict the number of duty arcs in the paths by applying the algorithm several times on a sufficiently small subnetwork. We explain this concept with an example. Consider the weekly problem with the horizon of one week, and suppose that pairings cannot exceed 5 days

and M=5. In this case the mixed segment/duty timeline network spans the horizon of 12 days, 7 days for one week and 5 to account for pairings that wrap around. The nodes on the last 5 days correspond to the legs from the first 5 days except that they are shifted in time by one week. We first run the parametric longest path algorithm on the network that consists of only the nodes starting on days 1,2,3,4 and 5. This restricts the paths to the length of at most 4 duty arcs. The algorithm computes the values $\tau_{i,4}^{cb}$ for every leg i that starts on day 1, the values $\tau_{i,3}^{cb}$ for every leg i that starts on day 2, the values $\tau_{i,2}^{cb}$ for every leg i that starts on day 3, and the values $\tau_{i,1}^{cb}$ for every leg i that starts on day 4. Next we form the network consisting of only the nodes on days 2,3,4,5 and 6 and we again run the parametric longest path algorithm. Now we obtain the values $\tau_{i,4}^{cb}$ for every leg i that starts on day 2, the values $\tau_{i,3}^{cb}$ for every leg i that starts on day 3, and so forth. The procedure is then repeated until we obtain all $\tau_{i,j}^{cb}$.

Whenever a pairing with score less than 1 is found, it is stored in the pool. The pool has a maximum size of 100,000. When the pool reaches its maximum size, we select 20,000 pairings with the lowest score that remain in the pool and all other pairings are discarded. When pricing is finished, we add to the subproblem 20,000 pairings from the pool with the smallest score. If at the end the pool has less than 20,000 pairings, then we add all the pairings from the pool. When the subproblem has more than 50,000 columns, we remove from the subproblem all the nonbasic columns.

To obtain an initial feasible solution, we greedily select a given number of pairings and solve the LP over these pairings. In all of the problems this heuristic yields a fesible LP.

3.2 Computational Results

The computational experiments were conducted on a PC with a Pentium III 1 GHz processor, 256 MB of main memory and the Windows 2000 operating system. The algorithms were implemented in Visual C++ version 6.0 and we used the linear programming solver CPLEX version 7.0.

We present the computational results for solving the LP relaxation of airline crew scheduling instances. We have formed 14 instances and for each instance we created two different problems. In one problem we consider the computationally difficult 8-in-24 rule and the other problem does not assume this rule. The latter problems are denoted by primes, e.g. the problem P1 assumes the 8-in-24 rule whereas the problem P1' is the identical problem except that it does not have this rule. The problems with 8-in-24 rule mimic a typical U.S. domestic crew scheduling problem. On the other hand, the 8-in-24 rule is not present in long-haul problems and it does not necessarily apply in non U.S. countries and therefore the problems without the 8-in-24 rule reflect these scenarios. Table 1 shows the problem type, the size of the instances, and the total number of pairings. By the number of legs we refer to the number of different flights without taking into account their repetition in the time horizon. The column '# of rows' shows the number of rows in the LP relaxation. Note that if we do not consider the 8-in-24 rule, then we get up to three times as many pairings. All of the instances were created from a weekly flight schedule of a major US airline. All the feasibility rules, cost structure and parameters are identical to those used by the airline. The flight network has a hub-and-

spoke structure, i.e. there are a few stations with heavy flight activity and all the remaining stations have flights mostly to these high activity stations, and every instance has 6 crew bases.

	type	# of legs	# of rows	# of pairings with 8-in-24	# of pairings without 8-in-24
P1	daily	149	149	1,393,897	3,607,004
P2	daily	175	175	2,256,809	5,596,648
P3	weekly	62	441	7,827,689	12,930,660
P4	weekly	123	861	6,201,276	13,070,302
P5	weekly	137	959	7,493,572	16,263,593
P6	weekly	135	945	21,431,998	43,683,246
P7	dated	337	614	22,056,040	26,792,753
P8	dated	71	449	14,707,867	20,523,678
P9	dated	142	459	15,357,341	21,637,974
P10	dated	353	630	23,448,011	28,399,866
P11	daily	188	188	1,542,496	6,197,746
P12	dated	78	479	4,456,333	8,345,789
P13	daily	194	194	6,535,998	20,778,195
P14	daily	174	174	3,451,078	11,162,473

Table 1. Size and number of pairings

From these 28 problems we have selected 10 diversified problems to carry out a detailed analysis of the pruning rules. Table 2 shows the impact of the approximate stage 1. We compare the algorithm without any pruning, i.e. in each iteration we enumerate all of the pairings, with the algorithm where we use the approximate stage 1 followed by no pruning at all. All times are CPU times in minutes. For the runs with no pruning, we report the number of iterations, the total execution time and the time of the pairing enumeration function, which is given in the column 'per iter'. For the approximate stage 1 pruning, we first report the number of approximate iterations in the column 'appr' under '# iter', i.e. the number of iterations before we switch to complete enumeration, and the total number of iterations. Next we give the average time of pairing generation in the approximate stage 1 (column 'appr' under 'time') and the total execution time. The last two columns show the improvement as a percentage with respect to the running time and the objective value after the approximate stage 1. The objective value improvement is defined as $100(z_I - \bar{z})/(z_I - z_{LP})$, where z_{LP} is the optimal LP value, $\bar{z}$ is the objective value after the approximate stage 1, and z_I is the initial LP value. We observe that for half of the problems the approximate pruning rules prune all the pairings with score less than 1 and therefore have no impact. For three problems the solution after the approximate stage 1 is really close to the optimal solution, which results also in a lower overall execution time. The average pairing generation time in the approximate stage 1 is up to 10 times lower than the time to enumerate all of the pairings. Therefore the approximate rules 1 and 2 achieve the goal of substantially reducing the enumeration time but however sometimes they tend to prune too much. Note that for problems P3' and P4' the objective value after the approximate stage 1 is close to the optimal value however many more iterations are required to reach optimality.

	no pruning			approximate stage 1					
	# iter	time		# iter		time		improvement (%)	
		total	per iter	appr	total	appr	total	time	obj
P1'	7	11	1.5	11	13	0.25	7.5	32	88
P2	7	23	3.2	1	7	0	23	0	0
P2'	7	20	2.8	2	8	1	18	10	1
P3'	10	23	2	11	16	0.2	16	30	99
P4	10	58	4.9	4	12	1	56	3.4	11
P4'	22	58	2	14	31	0.5	55	5	90
P5	13	102	6.9	1	13	0	102	0	0
P7	10	354	35	2	10	0	354	0	0
P8	8	96	12	1	8	0	96	0	0
P9	7	99	12.4	2	8	1	76	23	0

Table 2. Computational results for approximate stage 1

Next we consider the approximate stage 2. The computational results are given in Table 3. The table columns corresponding to the approximate stage 2 have identical meaning as the corresponding columns in Table 2. Here by running only the approximate stage 2 we mean that we apply the approximate stage 2 pruning rules and then we switch to a complete enumeration without pruning. Table 3 also gives results for the algorithm that uses both approximate stages 1 and 2. This algorithm performs first the approximate stage 1, then it switches to the approximate stage 2, and when the approximate pruning rules 3 and 4 prune all pairings with score less than 1, a complete enumeration is used. The improvements in time and in the objective value are with respect to the runs with no pruning reported in Table 2. Columns 'appr 1' and 'appr 2' under '# iter' show the number of iterations of the approximate stages 1 and 2, respectively.

	approximate stage 2						approximate stages 1 and 2					
	# iter		time		improvement (%)		# iter			time	improvement (%)	
	appr	total	appr	total	time	obj	appr 1	appr 2	total		time	obj
P1'	6	10	0	5.5	50	76	11	5	18	8	27	96
P2	5	10	0	16.3	29	64	1	6	11	16.5	29	64
P2'	7	10	0	7.8	61	85	2	7	14	17.3	13.5	71.6
P3'	4	7	0	7	70	99	11	2	15	10.5	54	99
P4	16	24	0	48.5	16.4	92	4	11	22	50	14	84.5
P4'	9	32	0	76	0	89	14	6	30	43	26	95
P5	8	14	0	48.5	16.4	85	1	8	15	50	16.4	85
P7	18	25	0.5	328	7	49	2	26	33	233	34	49
P8	6	9	0	36	63	99.5	1	6	10	36	63	99.5
P9	7	10	0	41.5	58	99.8	2	9	13	30	70	99.8

Table 3. The impact of the approximate stage 2

From Table 3 we conclude that the approximate stage 2 alone leads to the running time improvements from 7 to 70 percent, except for the P4' problem. The average

running time improvement is 37%. Problem P4' is interesting since using only the approximate stage 1 we obtain a substantial improvement in the objective value however the approximate stage 2 prunes all the pairings and therefore does not yield any improvement. With the exception of P4', the approximate pruning rules 3 and 4 yield better results. The time per pairing generation is extremely low in the approximate stage 2, typically just a few seconds. By combining the two approximate stages we achieve a reduction in total time for every problem. The achieved reductions are between 14 and 70 percent. The combined strategy is a clear winner for problems P4', P7, and P9, and it is outperformed by the approximate stage 2 alone in all other problems. In P4', in the algorithm that combines the two approximate stages, the initial dual vector for the approximate stage 2 is different from the initial dual vector if the approximate stage 2 alone is used and therefore we have 6 approximate stage 2 iterations in the combined algorithm. It is clear that the best strategy is instance dependant but the combined strategy has the best performance over all the instances. The average improvement in the running time is 35%. It is interesting to note that for some of the instances the objective value after the approximate stages 1 and 2 is close to the optimal value however the algorithm still needs a significant number of iteration to reach optimality. This has already been observed in Table 2.

In Table 4 we compare the algorithm with the approximate stages 1 and 2 followed by a complete enumeration with the algorithm that uses the approximate stages 1 and 2 and then the exact pruning rules. Columns 'per iter' show the average pairing enumeration time in iterations that follow the approximate stages. This time is significantly lower when using exact pruning, which shows that the exact pruning rules are beneficial. We give in the column 'comp τ' the average per iteration time of running the primal-dual algorithm for computing the values τ used in the exact pruning rules 2 and 3. These times are reasonably low and the gains from pruning are substantially larger. The time improvement values are with respect to the algorithm that uses only the two approximate stages followed by complete enumeration of pairings. Employing all the pruning rules improves the running time by 30 to 85 percent. The last 3 columns show the number of prunings by the 3 exact pruning rules. Note that the exact pruning rules 1 and 2 prune substantially more pairings than the exact pruning rule 3. This is not surprising since the exact pruning rule 3 does not use the factor $\sum_{d \in p} dc_d \Big/ \sum_{d \in p} dfl_d$ and therefore it seems to be weak. Due to different dual vectors that are generated by the algorithms in the exact stage, for some problems the number of iterations in the algorithm without exact pruning exceeds the number of iterations in the combined algorithm. We call the algorithm that uses both approximate stages and then the exact algorithm the *approximate*/*exact* algorithm.

To study the impact of the approximate stages 1 and 2 we have performed computational experiments with a variant of the algorithm that uses only exact pruning in every iteration. The results are given in Table 5. The next to last column shows the improvement of the running time with respect to the algorithm without any pruning. The last column compares the execution time of the approximate/exact algorithm with the algorithm that uses only exact pruning. In all of the instances the former algorithm outperforms the latter algorithm, the average improvement being 47%. This shows that the approximate stages are beneficial.

	appr stages 1 and 2			approximate stages 1, 2 and exact							
	# iter	time		# iter	time			improvement in time (%)	# prunings		
		total	per iter		total	per iter	comp τ		Rule 1	rule 2	rule 3
P1'	18	8	1.5	12	3.9	0.5	0.2	51	119,397	397,572	537
P2	11	16.5	3.2	10	4.5	0.3	0.25	72	163,454	377,157	662
P2'	14	17.3	2.8	15	5.9	0.7	0.25	65.9	185,980	688,071	12,744
P3'	15	10.5	2	15	7	0.9	0.05	33.3	344,787	341,847	2,233
P4	22	50	4.9	24	25.3	1	0.25	49.4	272,716	232,907	3,809
P4'	30	43	2	17	29.2	1	0.25	32	336,437	217,130	463
P5	15	50	6.9	15	20	1.5	0.3	60	413,186	307,794	4,345
P7	33	233	35	20	33.5	3.7	0.08	85.6	2,442,275	873,422	52,995
P8	10	36	12	10	7.2	2	0.03	80	776,123	439,280	77
P9	13	30	12.4	8	6	2.1	0.03	80	841,247	275,400	1,389

Table 4. Computational experiments with exact pruning

	only exact pruning				
	# iter	Time		vs. no pruning (%)	vs. approx. /exact (%)
		total	per iter		
P1'	10	6.4	1.6	42	-39
P2	11	7.3	1.5	68	-38
P2'	11	13.5	0.8	33	-56
P3'	10	12.3	0.8	47	-43
P4	9	35	0.3	40	-28
P4'	20	38	0.5	34	-23
P5	13	58	0.2	43	-66
P7	13	54	0.2	85	-38
P8	10	22	0.5	77	-67
P9	13	25.5	0.5	74	-76

Table 5. The impact of the approximate stages 1 and 2

In Table 6 and Table 7 we present the computational results for all of the problems. Table 6 shows the results for the problems with the 8-in-24 rule and Table 7 without this rule. Note that for approximately half of the problems the computational times are larger for the problems with the 8-in-24 rule despite the lower number of pairings, which shows that this rule is indeed computationally expensive. We compare the execution time of the approximate/exact algorithm and the implementation that does not use any pruning. The results show that by using the pruning rules we reduce the overall running time by 56 to 94 percent for the problems with the 8-in-24 rule and the average reduction is 82%. For the problems without the 8-in-24 rule the decrease in the execution time ranges from 49 to 86 percent and the average reduction is 66%. The runs with the pruning rules have more iterations due to the approximate stages but however the time per iteration is significantly lower. The higher reduction for the problems with the 8-in-24 rule is expected since pruning an operation that is computationally intense is more effective.

	no pruning			approximate/exact			
	# iter	Time		# iter	Time		improvement in time (%)
		total	per iter		total	per iter	
P1	7	11.5	1.6	11	2.5	0.2	78
P2	7	23	3.2	10	4.5	0.4	80.4
P3	6	42.7	6.8	10	4.5	0.4	89.4
P4	10	58	4.9	24	25.3	1	56.3
P5	13	102	6.9	15	20	1.3	80.4
P6	17	345	19.5	20	61	3	82.3
P7	10	354	35	20	33.5	1.7	90.5
P8	8	96	12	10	7.2	0.7	92.5
P9	7	99	12.4	8	6	0.7	94
P10	10	397	39	15	31	2	92.2
P11	9	66	7.3	15	23	1.5	77.3
P12	9	146	17	16	35	2.1	76
P13	11	107	9.7	18	33	1.8	69
P14	11	82.5	7.5	12	10.6	0.9	87.5

Table 6. Computational results for the problems with the 8-in-24 rule

	No pruning			approximate/exact			
	# iter	time		# iter	Time		Improvement in time (%)
		total	per iter		total	per iter	
P1'	7	11	1.5	12	3.9	0.32	64.5
P2'	7	20	2.8	15	5.9	0.4	70.5
P3'	10	23	2	15	7	0.47	69.6
P4'	22	58	2	17	29.2	1.7	49.6
P5'	21	73.5	2.6	24	35	1.4	52.3
P6'	20	158	6.8	24	68	2.8	57
P7'	18	158.5	9	27	39	1.4	75.4
P8'	9	40.8	4.2	14	14.5	1	64.5
P9'	10	47.8	4.5	15	15.2	1	68.2
P10'	17	173	10	28	37.5	1.3	78.3
P11'	14	98	7	14	13.5	1	86.2
P12'	11	57	5.1	16	21	1.3	63
P13'	17	289	17	17	82	4.8	71.6
P14'	14	92.5	6.6	17	40.5	2.4	56

Table 7. Computational results for the problems without the 8-in-24 rule

4 Concluding Remarks

Our pairing cost structure consists of a maximum of three linear terms. Some airlines add to the cost structure an additional linear term, e.g. fixed cost per overnight connection arc to reflect the crew cost of lodging and meals. Our pruning rules can be easily extended to such a case.

The discussion of the pruning rules revolves around the pricing algorithm that is based on enumeration. The presented framework is applicable to constrained shortest

path algorithms as well. Suppose that during the execution of a constrained shortest path algorithm we pick a label vector associated with duty d from the priority queue. The algorithm next scans all the outgoing connection arcs from d and for each arc we compute the new label vector r and we insert it in the priority queue. Due the nature of the constrained shortest path algorithms, given r it is possible to reconstruct the path q from the source to the duty corresponding to r. Note that to be able to use the pruning rules, we need $\sum_{d \in \tilde{d}(q)} dfl_d$ and $\sum_{d \in \tilde{d}(q)} dy_d$, which are easily computable. Given these two quantities, we can check if one of the pruning conditions applies. If yes, then we do not add the new resource vector r to the priority queue. Typically $\sum_{d \in \tilde{d}(q)} dfl_d$ and $\sum_{d \in \tilde{d}(q)} dy_d$ correspond to two labels and therefore in this case there is no extra computation involved. Our pruning rules and the rational behind constrained shortest path algorithms are complimentary. We fathom a partial pairing if its 'tail' is bad, however, the latter algorithms detect via dominance in the label vectors if the partial pairing is bad. Our pruning rules look ahead while constrained shortest path algorithms look back if there were already a better partial pairing with respect to the labels.

Another application of our pruning rules is in primal-dual algorithms, Hu and Johnson (1999), and steepest edge algorithms, Gopalakrishnan et. al. (2001). In every iteration both algorithms maintain a dual feasible solution π and a vector ρ, which is not necessarily dual feasible. The former algorithm seeks a nonnegative scalar t such that $(1-t)\pi + t\rho$ is a dual feasible vector with the largest objective value. In the latter algorithm a nonnegative scalar t is sought such that $\pi + t\rho$ is a dual feasible vector with the largest optimal value. It is easy to see that t corresponds to

$$\min_p \left\{ \left(c_p - \sum_{i \in p} \pi_i \right) \Big/ \sum_{i \in p} (\rho_i - \pi_i) \,|\, \sum_{i \in p} (\rho_i - \pi_i) > 0 \right\}, \min_p \left\{ \left(c_p - \sum_{i \in p} \pi_i \right) \Big/ \sum_{i \in p} \rho_i \,|\, \sum_{i \in p} \rho_i > 0 \right\},$$

respectively. Since our score has the same structure, the pruning rules can easily be adapted to this case as well. One change that needs to be done is to replace the cutoff value of 1 in the rules with the best value of t found so far.

References

ANBIL, R., JOHNSON, L. and TANGA, R. A global optimization approach to crew scheduling. *IBM Systems Journal* **31**, 62-74 (1991).

ANBIL, R., FORREST, J. and PULLEYBLANK, W. 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 (1998).

ANBIL, R., GELMAN, E., PATTY, B., and TANGA, R. Recent advances in crew pairing optimization at American Airlines. *Interfaces* **21**, 62-74 (1991).

ANDERSON, E., HOUSOS, E., KOHL, N. and WEDELIN, D. Crew pairing optimization. *Operations Research in the Airline Industry*, 228-258, Yu G. (editor), Kluwer (1998).

BARNHART, C., JOHNSON, E., NEMHAUSER, G., SAVELSBERGH, M. and VANCE, P. Branch-and-Price: Column generation for solving huge integer programs. *Operations Research* **46**, 316-329 (1998).

BARNHART, C., JOHNSON, E., NEMHAUSER, G. and VANCE, P. Crew scheduling. *Handbook of Transportation Science*, 493-521, Hall R. (editor), Kluwer (1999).

BIXBY, R., GREGORY, J., LUSTIG, I., MARSTEN, R. and SHANNO, D. Very large-scale linear programming: a case study in combining interior point and simplex methods. *Operations Research* **40**, 885-897 (1992).

DESAULNIERS, G., DESROSIERS, J., IOACHIM, I., SOLOMON, M. and SOUMIS, F. A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. *Fleet Management and Logistics*, 57-93, Crainic T. and Laporte G. (editors), Kluwer (1998).

DESROCHERS, M. and SOUMIS, F. A generalized permanent labeling algorithm for the shortest path problem with time windows. *INFOR* **26**, 191-212 (1988).

DESROSIERS, J., DUMAS, Y., DESROCHERS, M., SOUMIS, F., SANSO, B. and TRUDEAU, P. A breakthrough in airline crew scheduling. *Technical Report G-91-11, GERAD* (1991).

DESROSIERS, J., DUMAS, Y., SOLOMON, M. and SOUMIS, F. Time constrained routing and scheduling. *Handbook in Operations Research/Management Science, Network Routing* **8**, 35-139, Ball M., Magnanti T., Monma C. and Nehmhauser G. (editors), Elsevier Science (1995).

GERSHKOFF, I. Optimizing flight crew schedules. *Interfaces* **19**, 29-43 (1989).

GOPALAKRISHNAN, B., JOHNSON, E., LEE, E. and PRITCHETT, A. A subproblem approach for solving the airline crew pairing problem. Presentation at INFORMS Fall 2001, Miami Beach (2001).

HU, J. and JOHNSON, E. Computational results with a primal-dual subproblem simplex method. *Operations Research Letters* **25**, 149-158 (1999).

KLABJAN, D., JOHNSON, E. and NEMHAUSER, G. Solving large airline crew scheduling problems: Random pairing generation and strong branching. *Computational Optimization and Applications* **20**, 73-91 (2001).

MAKRI, A. and KLABJAN, D. Algorithms for the source-to-all maximum cost to time ratio problem on acyclic networks. *Working Paper*, University of Illinois at Urbana-Champaign (2001). Available from http://www.staff.uiuc.edu/~klabjan/professional.html.

VANCE, P., ATAMTURK, A., BARNHART, C., GELMAN, E., JOHNSON, E., KRISHNA, A., MAHIDHARA, D., NEMHAUSER, G. and REBELLO, R. A heuristic branch-and-price approach for the airline crew pairing problem. *Technical Report LEC-97-06*, Georgia Institute of Technology (1997).