Technical University of Denmark

Subsequence Generation for the Airline Crew Pairing Problem

Rasmussen, Matias Sevel; Lusby, Richard Martin ; Ryan, David; Larsen, Jesper

Publication date:
2011

Document Version
Publisher's PDF, also known as Version of record

Subsequence Generation for the Airline Crew Pairing Problem

Report 9 2011

DTU Management Engineering

Matias S. Rasmussen
Richard M. Lusby
David M. Ryan
Jesper Larsen

May 2011

DTU Management Engineering
Department of Management Engineering

Subsequence Generation for the Airline Crew Pairing Problem

Matias Sevel Rasmussen[1], Richard M. Lusby[1], David M. Ryan[2], and Jesper Larsen[*,1]

[1]Department of Management Engineering, Technical University of Denmark, Denmark
[2]Department of Engineering Science, The University of Auckland, New Zealand

May, 2011

Abstract

Good and fast solutions to the airline crew pairing problem are highly interesting for the airline industry, as crew costs are the biggest expenditure after fuel for an airline. The crew pairing problem is typically modelled as a set partitioning problem and solved by column generation. However, the extremely large number of possible columns naturally has an impact on the solution time.

In the solution method of this work we severely limit the number of allowed subsequent flights, i.e. the subsequences, thereby significantly decreasing the number of possible columns. Set partitioning problems with limited subsequence counts are known to be easier to solve, resulting in a decrease in solution time.

The problem though, is that a small number of deep subsequences might be needed for an optimal or near-optimal solution and these might not have been included by the subsequence limitation. Therefore, we try to identify or generate such subsequences that potentially can improve the solution value.

We benchmark the subsequence generation approach against a classical column generation approach on real-life test instances. We consider the LP relaxation and compare the quality and the integrality of the solutions. The LP solutions from the subsequence generation approach are less fractional, but it comes at the cost of a worse solution quality.

[*]Corresponding author: E-mail: jesla@man.dtu.dk. Address: Department of Management Engineering, Technical University of Denmark, Produktionstorvet, Building 424, DK-2800 Kgs. Lyngby, Denmark. Tel.: +45-45253385. Fax: +45-45933435.

The approach in the present paper is novel. To our knowledge generation of subsequences have not been described and tested previously in the literature.

Keywords: Airline crew pairing, Subsequence generation, Column generation, Limited subsequence, Crew scheduling, Real-life application, Set partitioning, LP relaxation

1 Introduction

Crew costs are the second largest expense for an airline company. Only fuel costs are higher, see Gopalakrishnan and Johnson (2005). Here it is also reported that, for instance, American Airlines spent USD 1.3 billion on crew in 1991. The expenditures for an airline can roughly be divided equally between three areas: Fuel, crew, and other costs (buildings, maintenance, administrative staff, etc.). As fuel costs cannot be controlled by an airline, crew costs are probably the most important area for potential savings. Therefore, airline crew scheduling has received a lot of attention in the literature, and consequently, optimisation is heavily used by the airlines. With such a large amount of money being spent on crew, even small improvements in how they are scheduled can result in significant savings.

The airline crew pairing problem which is dealt with in this work is a part of a larger series of optimisation problems, see Figure 1. The times are for Air New Zealand's domestic scheduling. The first step is *flight timetabling*. In this step a schedule of all the flights that the airline will fly is constructed. The next steps are *fleet assignment*, where aircraft types are allocated to the flights, and *aircraft routing*, where the aircraft routes are laid. These steps, however, do not directly influence the crew scheduling. The crew pairing step (which is the focus of this paper) finds sequences of flights that can be flown in a feasible way at a minimum cost. These sequences of flights are called *pairings* and are anonymous, that is they are not associated with a specific crew member. The crew pairing problem can be solved separately for cockpit crew and cabin crew, and it is also solved separately per aircraft type qualification. The last step is *crew rostering* where pairings are combined to form actual rosters for individual crew member. The crew pairing and the crew rostering steps are together called *airline crew scheduling*.

This paper presents a novel *subsequence generation* approach to solving the crew pairing problem. The subsequence generation approach is to our knowledge not found elsewhere in the literature. We consider the linear programming (LP) relaxation of a set partitioning formulation of the problem. The idea is to generate subsequences of flights that appears in the optimal pairings instead of—as in classic column generation—generating the actual pairings. Whenever a subsequence is found, that is generated, a whole set of

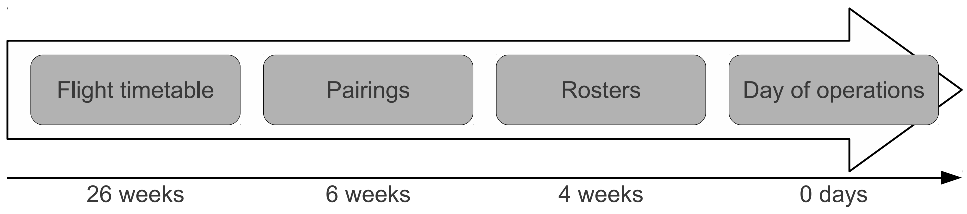


Figure 1: The airline crew scheduling process.

pairings containing that subsequence is enumerated and added to the LP relaxation. The devised solution algorithm is tested on real-life data instances and benchmarked against classic column generation.

In Gopalakrishnan and Johnson (2005) a recent survey of airline crew scheduling can be found. The authors describe the different approaches that have been used over the last two decades, and point out promising directions for future work in the area. The crew pairing problem is treated separately and in detail. Barnhart et al. (2003) give a text book description of airline crew scheduling and also have a detailed section on crew pairing with examples. They formulate the crew pairing problem as a set partitioning problem and describe how the problem can be solved as a weekly problem or a dated problem. The weekly problem approach exploits repetitive patterns of flights over the weekdays, and is thus able to break the problem into smaller parts, which are then combined. This division of the problem is of course a trade-off against optimality. The dated problem approach on the other hand solves the problem directly, and is necessary for flight timetables where flights are not repeated several times a week. The complex cost structures for pairings are described by Gopalakrishnan and Johnson (2005) and Barnhart et al. (2003). Andersson et al. (1998) describe different approaches to crew pairing and give a detailed introduction to the Carmen (now Jeppesen) system for solving the crew pairing problem. The Carmen system uses a priori column generation; however, it has separated the checking of the pairing requirements into a special rules language. The Carmen system uses the algorithm described by Wedelin (1995). Desaulniers et al. (1998) present the crew pairing model as a special case of a generic air crew scheduling model, that also covers, for instance, rostering. They solve the crew pairing problem with column generation. AhmadBeygi et al. (2009) develop an integer programming model for generating pairings. The model can be used especially in research to overcome the time-consuming task of implementing a pairing generator. Butchers et al. (2001) describe airline optimisation problems in general and the crew pairing problem in particular for Air New Zealand's domestic and international schedule. Also here the crew pairing problem is formulated as a set partitioning problem. Lavoie et al. (1988)

use a set covering formulation and perform column generation on a duty period network. A duty period is a sequence of flights that corresponds to a day's work, see more in Section 2. Graves et al. (1993) use a set partitioning formulation and do column generation on a network of flights. Vance et al. (1997) use a two-stage approach. First flights are combined to form duty periods, and next duty periods are combined to form pairings. Using dynamic constraint aggregation crew scheduling can be solved in an integrated approach, see Saddoune et al. (2011). In this way all constraints are virtually present in the master problem, but in an aggregated form, where basically constraints belonging to the same pairing are just represented by one active constraint. The update of these active constraints lead to a complex setup in the interplay with the column generator. This, though, does at present remain a very complex and time-consuming approach limited to academic environments only.

The remainder of this paper is organised as follows. In Section 2, we present a formal definition of the airline crew pairing problem. In Section 3, we introduce the concept of subsequence limitation and the motivation behind it. In Section 4, we develop the suggested subsequence generation solution algorithm. In Section 5, we present real-life test instances, and we show benchmark results from the comparison between the subsequence generation approach and a classical column generation approach. Finally, in Section 6, we conclude on the work and point out directions for future research.

2 Problem formulation

Let $\mathcal{F}$ denote the set of flights in the flight schedule for an airline. A *duty period* is a sequence of flights from $\mathcal{F}$ which can be flown by an anonymous crew member. A duty period must comply with several rules and regulations in order to be feasible. A crew member can either be *operating* or *passengering* (sometimes called *deadheading*) on a flight. Passengering allows crew members to be repositioned in order to operate other flights. A duty period consists of *flying time*, where the crew member is operating the flight, and *idle time*, which together give the *elapsed time*. Each duty period has a maximum flying time and a maximum elapsed time, as well as a maximum number of flights that can be operated. Duty periods must also respect meal break regulations. Duty periods are separated by *rest periods*, which must have a minimum length. Starting and ending a duty period impose a *sign-on* and *sign-off* time, respectively.

A *pairing* (sometimes called a *tour-of-duty*) is a sequence of duty periods and rest periods. Every airline has a set of *crew bases*, i.e. airports from where crew can start working. A feasible pairing must start and end at the same crew base. Pairings can only contain up to a maximum number

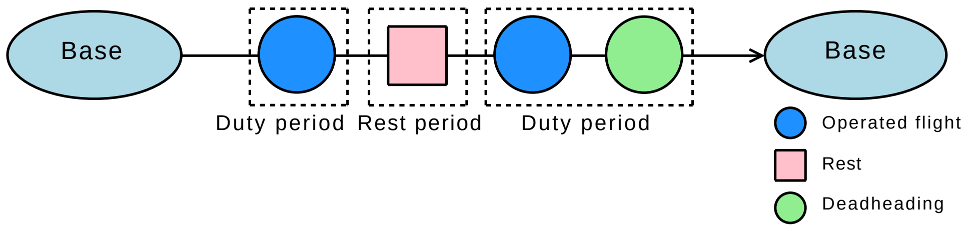


Figure 2: Illustration of a pairing.

of duties, and a pairing is only allowed to stretch over a certain number of *mandays*. The manday count is increased every time midnight is passed in the time zone where the pairing originates. Different airlines use different and quite complex ways of calculating the cost of a pairing, for examples of this see Gopalakrishnan and Johnson (2005) and Barnhart et al. (2003). For the research carried out in the present paper, we use the pairing's idle time as the cost of the pairing. That way crew utilisation is maximised. An illustration of a pairing can be seen on Figure 2.

The *airline crew pairing problem* is then to find the set of pairings that covers all flights exactly once at minimum cost. Let $\mathcal{P}$ be the set of feasible pairings. The problem is modelled as a *set partitioning problem*. Each row corresponds to a flight and each column corresponds to a pairing. Let $\bar{m} = |\mathcal{F}|$ be the number of flights and $n = |\mathcal{P}|$ be the number of pairings. Now, the pairings can be represented by a binary $\bar{m} \times n$ matrix $\boldsymbol{A}$, where the entries are defined by $a_{ij} = 1$ if flight $i \in \{1, \ldots, \bar{m}\}$ is contained in pairing $j \in \{1, \ldots, n\}$, and $a_{ij} = 0$ otherwise. Let c_j be the cost of pairing $j \in \{1, \ldots, n\}$. The decision variables x_j for $j \in \{1, \ldots, n\}$ govern the inclusion of pairing j in the solution and are binary. The mathematical programme can then be written as

$$\begin{array}{ll} \text{minimise} & \boldsymbol{c}^\top \boldsymbol{x} \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{1} \\ & \boldsymbol{x} \in \{0,1\}^n \ . \end{array}$$

Most airlines, however, extend this standard model to include the so-called *base constraints*. These constraints are required for distributing the pairings amongst the crew bases in a way that matches the actual distribution of where the crew is located geographically. Base constraints can be defined in many different ways. In order to simplify matters, we have chosen to include only one type of base constraint. A base constraint puts a lower or an upper bound on the number of mandays that can be worked out of a set of crew bases in a given time period. A pairing contributes to a base constraint, if the pairing origins from that set of crew bases in the specified time period.

The pairing's contribution to the base constraint is the manday count of the pairing and given as d_j, where $j \in \{1, \ldots, n\}$.

Let $\mathcal{B}$ denote the set of base constraints and set $m = \bar{m} + |\mathcal{B}|$. We can then augment $\boldsymbol{A}$ to an $m \times n$ matrix where

$$a_{ij} = \begin{cases} d_j & \text{if pairing } j \text{ originates from the set of crew bases and in the} \\ & \text{time period specified by base constraint } i, \\ 0 & \text{otherwise} \end{cases}$$

for $i \in \{m' + 1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$. The base constraints are of less-than-or-equal or greater-than-or-equal type, and most often have non-unit right hand sides. We therefore end up with a *generalised set partitioning model*, where slack and surplus columns are included to convert the inequality base constraints to equality constraints:

$$\begin{array}{ll} \text{minimise} & \boldsymbol{c}^\top \boldsymbol{x} \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \in \{0,1\}^n \ . \end{array}$$

Here, the flight set partitioning constraints for $i \in \{1, \ldots, \bar{m}\}$ have $b_i = 1$, and the base constraints for $i \in \{\bar{m} + 1, \ldots, m\}$ have $b_i \in \mathbb{Z}_+ \cup \{0\}$.

We allow for the possibility of leaving flights uncovered at a high objective value penalty, and we allow for the violation of base constraints, also with a high penalty. This is modelled by having feasibility singleton columns for flights and for base constraints in the model.

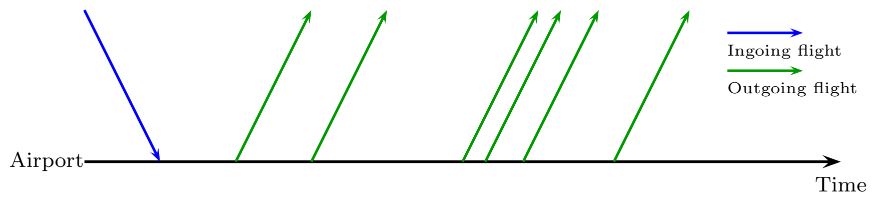
The number of possible pairings in the set partitioning formulation is very large, so the pairings are typically only enumerated implicitly by column generation. In the present approach we will, however, not perform column generation, but subsequence generation.

3 Subsequence limitation

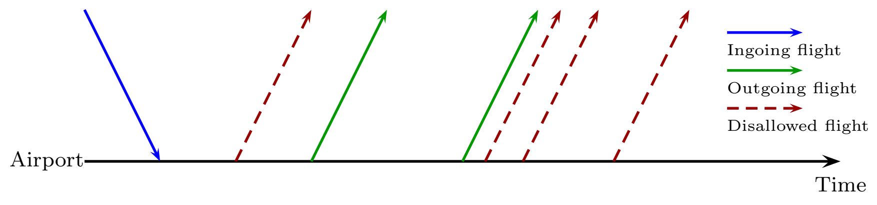
The *subsequences* for a flight $f \in \mathcal{F}$ are the set of pairs $(f, g) \in \mathcal{F}^2$ where $g \in \mathcal{F}$ is a subsequent flight that can follow f in a feasible way in a pairing. We denote this set $\mathcal{S}(f) \subset \mathcal{F}^2$. Subsequences are illustrated on Figure 3(a). In general terms for an $m \times n$ zero-one matrix $\boldsymbol{A}$ with entries a_{ij}, the subsequence set $\mathcal{S}(s)$, for any row s is given by

$$\mathcal{S}(s) = \{(s,t) : [\exists j \in \{1, \ldots, n\} : a_{sj} = 1, a_{ij} = 0 \text{ for } s < i < t, a_{tj} = 1]\} \ .$$

In the example on Figure 4 we have $\mathcal{S}(1) = \{(1,3), (1,4), (1,6)\}$. Matrices where the *subsequence count* $|\mathcal{S}(s)| \leq 1$ for all $s \in \{1, \ldots, m\}$ are said to have *unique subsequence*, and such matrices are balanced, see Ryan and Falkner (1988). Exploiting results from graph theory, see Conforti et al. (2001), we know that the LP relaxation of a set partitioning problem with a



(a) Subsequences.



(b) Severely limited subsequences.

Figure 3: Subsequences for an ingoing flight.

$$\begin{pmatrix} \boxed{1} & \boxed{1} & 1 & 1 & \boxed{1} & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \boxed{1} & 1 & 0 \\ \boxed{1} & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & \boxed{1} & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Figure 4: Subsequences for row 1.

balanced $\boldsymbol{A}$ matrix has an integral optimal solution. Intuitively, the closer we get towards unique subsequence, the closer we get to naturally integral LP solutions. Ryan and Falkner (1988) show experimental results to support this.

Therefore, we severely limit the subsequence count for each flight when generating pairings, see Figure 3(b). In this example the first disallowed flight is removed, because there is not enough ground time for a robust aircraft change. The three last disallowed flights are removed, because they have a lot of ground idle time, so it is not likely (though still possible) that they will end up in an optimal solution.

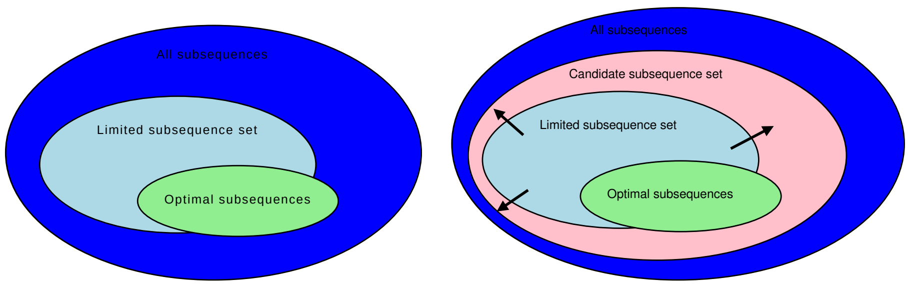
The possible subsequent outgoing flights for an ingoing flight f are now restricted to be in the *limited subsequence set* $\mathcal{L}(f) \subseteq \mathcal{S}(f)$. Let $\mathcal{S} = \bigcup_{f\in\mathcal{F}} \mathcal{S}(f)$ denote the set of all subsequences for all flights, let $\mathcal{L} = \bigcup_{f\in\mathcal{F}} \mathcal{L}(f)$ denote the set of limited subsequences for all flights, and let $\mathcal{O}$ denote an optimal subsequence set, i.e. an optimal solution, for all flights. Naturally, $\mathcal{O}$ has unique subsequence due to the set partitioning constraints. The relations between these three sets can be illustrated by a Venn diagram, see Figure 5(a). There could, of course, be more than one set of optimal subsequences, but we only show one set on the figure.

The disadvantage of this limited subsequence approach is that some optimal subsequences might be excluded. However, the approach results in significantly fewer possible pairings, and therefore a total enumeration of the pairings in $\mathcal{L}$ can be carried out. Moreover, when the LP relaxation is solved, fewer fractions are expected, as the subsequence count of all flights per construction is low.

4 Subsequence generation

To remedy the possible lack of optimal subsequences, the limited subsequence set is made to be dynamic. The core idea is to generate subsequences that will decrease the objective value. We exploit the fact, that in crew pairing the chosen subsequences will most often be close in time, which is natural, keeping the pairing cost definition in mind. Therefore, the hope is that only relatively few subsequences with much idle time have to be generated.

A *candidate subsequence set* $\mathcal{C}(f)$ is defined for all flights $f \in \mathcal{F}$, and again we define $\mathcal{C} = \bigcup_{f\in\mathcal{F}} \mathcal{C}(f)$. We have $\mathcal{L}(f) \subseteq \mathcal{C}(f) \subseteq \mathcal{S}(f)$ for all $f \in \mathcal{F}$ and $\mathcal{L} \subseteq \mathcal{C} \subseteq \mathcal{S}$, which is shown in Figure 5(b). The idea is now to expand $\mathcal{L}$ with attractive subsequences from $\mathcal{C}$. A subsequence $s \in \mathcal{C}$ is attractive if it is likely that $s \in \mathcal{O}$, where again $\mathcal{O}$ is a set of optimal subsequences. Iteratively an *attractive subsequence set* $\mathcal{A} \subseteq \mathcal{C}$ is found and added to $\mathcal{L}$. Although Figure 5(b) shows the set of optimal subsequences to be contained in the candidate subsequence set, there is no guarantee for this.



(a) Optimal subsequences might be missing.   (b) Limited subsequence set is dynamic.

Figure 5: The relations between the set of all subsequences $\mathcal{S}$, the limited subsequence set $\mathcal{L}$, the candidate subsequence set $\mathcal{C}$, and an optimal subsequence set $\mathcal{O}$.

In Algorithm 1 the outline of the algorithm for solving the LP relaxation of the pairing problem can be seen.

Algorithm 1 Subsequence generation

```
1: Find an initial limited subsequence set $\mathcal{L}$
2: Enumerate all pairings over $\mathcal{L}$
3: Solve the LP relaxation on these pairings
4: while stop criteria not met do
5:     Based on the LP dual vector, identify a set of attractive subsequences $\mathcal{A} \subseteq \mathcal{C}$
6:     Enumerate pairings for each of the subsequences in $\mathcal{A}$
7:     Set $\mathcal{L} := \mathcal{L} \cup \mathcal{A}$
8:     Expand the LP relaxation with the enumerated pairings and re-solve
9: end while
```

The means that is used to identify attractive subsequences is the dual vector from the LP solution. The dual vector is passed on to a pairing generator that produces negative reduced cost columns on a the candidate subsequence set $\mathcal{C}$. The pairing generator is a resource constrained shortest path solver, which is run on subsequences from $\mathcal{C}$. The shortest path solver is a labelling algorithm, see for instance Irnich and Desaulniers (2005). The negative reduced cost columns, that are returned from the pairing generator, are analysed in order to collect statistics about the subsequences in $\mathcal{C} \backslash \mathcal{L}$.

The pairing generator is run sequentially on N different networks consisting of subsequences $\mathcal{C}^k \subseteq \mathcal{C}$ for $k \in \{1, \ldots, N\}$ with $\bigcup_{k=1}^{N} \mathcal{C}^k = \mathcal{C}$. The networks are kept small, so that the shortest path solver can execute very fast. In crew pairing there are four classes of subsequences that are very important to recognise:

1. *Follow-the-aircraft subsequences*: A follow-the-aircraft subsequence is a subsequence, where the crew flies out on the same aircraft as they flew in with. This type of subsequence is very robust towards possible delays, and one would expect the majority of the subsequences in an optimal crew pairing solution to be follow-the-aircraft. This expectation is supported by data from Air New Zealand. The follow-the-aircraft subsequence is unique, as there can only be one subsequent flight on the same aircraft. Most often the follow-the-aircraft subsequence will be low-cost, because the minimum sit time for crew is close to the minimum turnaround time for the aircraft. Being unique, robust, and low-cost, the follow-the-aircraft subsequence is the most attractive subsequence class.

2. *Robust subsequences*: A crew coming in on flight f can leave on flight g, if the *minimum sit time* is respected. However, if flight f is delayed and the time difference between arrival and departure of the two flights is exactly the minimum sit time, then flight g will also be delayed. A way to try avoid this delay propagation, is to add some *buffer time* to the minimum sit time. This of course comes at a higher pairing cost, as the crew might get unnecessary idle time. Studies in Ehrgott and Ryan (2002) show that delays increase during the day (and reset at midnight), so the buffer time should also increase during the day. We can now define a robust subsequence, as a subsequence, where the time difference between arrival and departure respects the buffer time needed at the given time of day.

3. *Meal break subsequences*: Naturally, crew is entitled to meal breaks, which is controlled by complex regulations. A meal break subsequence is a subsequence, where there is sufficient time for a meal break either inflight or on the ground between the flights.

4. *Overnight subsequences*: An overnight subsequence is a subsequence, where the time difference between arrival and departure is longer than the *minimum rest time.* An overnight subsequence is needed for a crew that flies in to a non-base airport late at night, where there is no subsequent flight to a home base. The overnight subsequence is also needed for a crew to fly out of a non-base airport early in the morning, where there has been no preceding incoming flight.

Follow-the-aircraft and robust subsequences are preferred from a pairing cost and robustness point of view, but the meal break and overnight subsequences are needed in order to make the pairings feasible and cover all flights. We will work with three subset of the candidate subsequence set $\mathcal{C}$ (so we have $N = 3$), based on these classes. The set $\mathcal{C}^1$ consists of follow-the-aircraft subsequences and robust subsequences. The set $\mathcal{C}^2$ consists of follow-the-aircraft subsequences and meal break subsequences, and $\mathcal{C}^3$ consists of follow-the-aircraft subsequences and overnight subsequences. The motivation for this setup, is, that we will now have networks for the pairing generator, that search specifically for robust, meal break, or overnight subsequences.

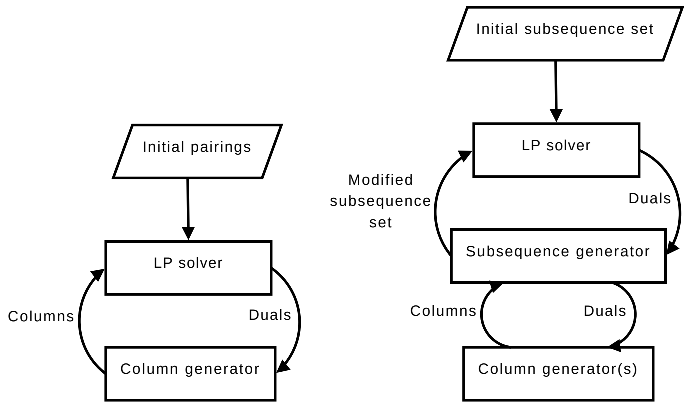
In order for the pairing generator to solve quickly, the subsequence count for all flights is kept low, that is $|\mathcal{C}^k(f)| \leq n^k$, where n^k is a small integer less than, say, five for all $k \in \{1, \dots, N\}$. It is important to note that n^k is an upper bound on the subsequence count in the given set for a flight. Consider for instance a flight going in to a non-busy airport. If the first robust outgoing flight (other than the follow-the-aircraft flight) departs, say, nine hours after the arrival of the ingoing flight, we do not include the flight as a robust subsequence, because it is unlikely that a pairing with such excessive idle time will end up in an optimal solution. Similar reasoning goes for the meal break and the overnight subsequence sets.

The set $\mathcal{C}^1$ is used as the initial limited subsequence set $\mathcal{L}$, where total enumeration is carried out.

For each subsequence $s \in \mathcal{C}$ we maintain four measures that are accumulated over all iterations and updated after analysis of the set of negative reduced cost columns returned by the pairing generator:

1. Count of columns containing s.
2. Count of different dual vectors that have produced columns containing s.
3. Sum of the reduced cost of columns that contain s.
4. Sum of the contribution from s to the negative reduced cost of columns containing s.

These measures can all be computed and updated quickly, which is important with respect to keeping the computational overhead of the approach at a minimum. The measures are correlated, so a high rank in one measure could also give a high rank in some of the other measures. In each iteration some subsequences are identified as attractive based on these four measures and added to $\mathcal{A}$. The goal is, of course, to be able to, as early as possible, identify the subsequences that potentially could end up in an optimal or near-optimal solution. Ideally one would identify subsequences that were non-dominated on all four measures and add these to $\mathcal{A}$. However, finding non-dominated points in four dimensions is very time-consuming, so instead we use just add one of the four measures, or we alternate between them. The idea of using dual information to identify attractive flights is also used by Barnhart et al. (1995). Here, only passengering flights are searched for, and added to a standard column generation approach.



(a) Classic column generation. (b) Subsequence generation.

Figure 6: Difference between classic column generation and subsequence generation.

Whenever a subsequence s is identified as an attractive subsequence, a whole set of columns which include the new subsequence is added to the LP relaxation. Enumeration is carried out in one of the following two ways:

1. Enumeration of all feasible pairings in $\mathcal{C}$ containing s.
2. Enumeration of all feasible pairings in $\mathcal{L}$ containing s.

The reason why a relatively large set of columns is added to the LP relaxation, is, that whenever a subsequence is identified as attractive, it is believed that it is likely to end up in an optimal solution. And hence, the optimal solution will contain one of the enumerated columns. The first way of enumerating gives rise to more columns in the LP relaxation. This is beneficial when it is strongly believed that a “right” subsequence is identified. The second way of enumerating is more restrictive on the set of columns that are added to the LP relaxation. Hence, the second enumeration scheme is expected to better at keeping the good integer properties of the initial limited subsequence set. With this scheme, adding a subsequence only increases the subsequence count with one for a single flight, namely for the first flight in the added subsequence. The subsequence generation algorithm is terminated, when the objective value has not improved significantly over a given span of iterations.

The differences between classic column generation and subsequence generation can be illustrated as the flowchart comparison in Figure 6. Subsequence generation has an extra part where columns are analysed.

	w08r01a	w08r01b	w08r01c	w08r01d	w08r01e	w08r02a	w08r02b	w08r02c	w08r02d	w08r02e	w08r03a	w08r03b	w08r03c	w08r03d	w08r03e	w08r04a	w08r04b	w08r04c	w08r04d
$\|\mathcal{F}\|$	450	430	430	370	400	380	400	320	450	350	320	420	420	450	320	400	400	350	320
$\|\mathcal{B}\|$	5	5	5	5	4	5	5	5	5	4	5	5	5	5	4	5	5	5	5

Table 1: Characteristics for the test instances. $|\mathcal{F}|$ is number of flights and $|\mathcal{B}|$ is number of base constraints.

5 Computational results

The goal in this section is to provide a benchmark analysis of the devised solution algorithm. We will benchmark the subsequence generation approach against a classical column generation approach. In this way, we will investigate the trade-off between solution quality and integrality of the solutions.

Air New Zealand have provided us with 19 real-life data instances from their domestic timetable. It should be noted that the domestic timetable for Air New Zealand covers Australia and various destinations in the Pacific Ocean. In order to allow the algorithm to terminate in reasonable time, we have limited the number of flights in the instances. Characteristics for these instances can be seen in Table 1.

We consider two quality measures for a solution. The first quality measure is the LP objective value. The objective value is value is the sum of idle time for all pairings plus penalties for leaving flights uncovered and penalties for violating base constraints. The second quality parameter is the number of uncovered flights. A flight is uncovered, if the feasibility column for that flight is chosen (perhaps fractionally) in the LP solution. Both quality parameters should be minimised.

To gauge the integrality of a solution, we consider two integer measures. Let $\boldsymbol{x}^* = (x_1^*, \ldots, x_n^*)^\top$ denote an LP solution to the crew pairing problem. The first integer measure counts the number of variables at value one in the solution and is calculated as $|\{i \in \{1,\ldots,n\} : x_i^* = 1\}| \cdot 100 / |\{i \in \{1,\ldots,n\} : x_i^* > 0\}|$. The second integer measure counts the number of "nice fraction"-variables in the solution, where a nice fraction is nonzero rational number smaller than or equal to one and a multiple of $1, 1/2, \ldots, 1/8$. It is calculated as $|\{i \in \{1,\ldots,n\} : x_i^* \in \{a/b : a, b \in \{1,\ldots,8\}\}\}| \cdot 100 / |\{i \in \{1,\ldots,n\} : x_i^* > 0\}|$. Integer measures should be maximised. The integer measure should give an indication of how easy or how difficult the fractions in the LP solution would be to resolve in a branch-and-bound framework. The use of such integer measures are based on the results from Ryan and Falkner (1988). The last measure we compare is run time.

We test different settings of the subsequence algorithm:

1. Maximum sizes for the candidate subsequence sets for a flight n^k: Experiments are carried out with n^k set to 2, 3, or 4.

2. Subsequence identification scheme: Either 1) count of columns, 2) count of different duals, 3) sum of reduced costs, 4) sum of subsequence contribution, or 5) an alternation of the previous is used.

3. Pairing enumeration scheme: Either enumeration is done over 1) $\mathcal{C}$ or 2) $\mathcal{L}$.

We identify one subsequence per iteration, and the algorithm is terminated, when the improvement in the LP objective value is less than 1% over a span of 100 iterations. The pairing generator returns up to eight columns with negative reduced cost each time it is run. For all of the 19 instances we run the algorithm for all 30 combinations of the settings described above. For all instances we also run a classic column generation algorithm where no limitation of subsequences is used. In order to make a fair comparison on quality, we set the robust buffer time to zero, so that we in effect do not search for robust subsequences. The time-out for all test runs is set to one hour, and all tests are run on 2.67 GHz Intel Xeon X5550 CPUs with 23.5 GB of memory. The algorithm is implemented in C++ and compiled with g++ 4.4.0 on a Linux computer. LP relaxations are solved with the LP solver from MOSEK version 6 using an academic license.

For all instances and for all 30 combinations of settings, we calculate ratios R/R_{ben}, where R denotes the value for the subsequence generation algorithm, we want to benchmark, and R_{ben} denotes the value for classic column generation to benchmark against. Table 2 shows the averages over all instances for each of the settings. For the solve time ratio it is preferable for our algorithm to have a ratio less than 1.00, meaning that subsequence generation is faster. For the root LP value ratio, and the uncovered ratio it is preferable for our algorithm to have a ratio as close to 1.00 as possible, as the optimal values from classic column generation is a lower bound. However, classic column generation times out on all instances, and therefore it would actually be possible to have a ratio less than 1.00. For the "at 1"-measure ratio and the "nice fraction"-measure ratio it is preferable for our algorithm to have a ratio larger than 1.00. From Table 2 it can be seen that the subsequence generation algorithm is always clearly faster than classic column generation. Classic column generation is still producing negative reduced cost columns at the one hour time-out limit. This is probably due to high degeneracy. From the table it can also be seen that the subsequence generation algorithm performs worse on the two quality parameters, LP objective value and number of uncovered flights. This was expected, as the subsequence generation algorithm is working on a limited subsequence set and therefore is more restricted than classic column generation, which has the full set of subsequences to choose from. Still, the conclusion that must be drawn from these averages, is, that more work on the subsequence identification procedure must be carried out, as there are some optimal subsequences missing.

Settings	Solve time ratio	Root LP value ratio	Uncovered ratio	“At 1”-measure ratio	“Nice fraction”-measure ratio
CG	1.00	1.00	1.00	1.00	1.00
2/1/1	0.00	1.77	1.63	1.95	1.98
2/1/2	0.00	1.84	1.80	1.83	2.02
2/2/1	0.00	1.75	1.66	1.75	2.11
2/2/2	0.00	1.84	1.82	1.57	1.94
2/3/1	0.00	1.76	1.73	1.34	1.81
2/3/2	0.00	1.85	1.80	1.47	1.90
2/4/1	0.00	1.75	1.64	2.10	2.10
2/4/2	0.00	1.90	1.87	1.47	1.90
2/5/1	0.00	1.75	1.74	1.76	1.92
2/5/2	0.00	1.83	1.79	1.92	2.05
3/1/1	0.01	1.25	1.25	1.22	2.08
3/1/2	0.00	1.30	1.21	1.48	2.08
3/2/1	0.01	1.25	1.27	1.06	1.89
3/2/2	0.00	1.30	1.30	1.43	1.89
3/3/1	0.01	1.25	1.29	0.92	1.74
3/3/2	0.00	1.28	1.23	1.43	1.89
3/4/1	0.00	1.25	1.35	1.13	2.06
3/4/2	0.00	1.29	1.31	1.98	2.04
3/5/1	0.01	1.25	1.32	1.09	1.85
3/5/2	0.00	1.28	1.34	1.47	1.94
4/1/1	0.01	1.23	1.22	1.07	1.71
4/1/2	0.01	1.26	1.21	1.72	1.57
4/2/1	0.01	1.23	1.24	1.02	1.52
4/2/2	0.01	1.26	1.28	1.49	1.73
4/3/1	0.01	1.24	1.27	0.88	1.80
4/3/2	0.01	1.24	1.30	1.26	1.80
4/4/1	0.01	1.21	1.26	0.81	1.51
4/4/2	0.00	1.25	1.26	1.62	2.06
4/5/1	0.01	1.23	1.31	0.80	1.55
4/5/2	0.01	1.24	1.21	1.44	1.75
2/*/*	0.00	1.81	1.75	1.72	1.97
3/*/*	0.00	1.27	1.29	1.32	1.95
4/*/*	0.01	1.24	1.25	1.21	1.70
/1/	0.01	1.44	1.39	1.55	1.91
/2/	0.01	1.44	1.43	1.39	1.85
/3/	0.01	1.44	1.44	1.22	1.82
/4/	0.00	1.44	1.45	1.52	1.95
/5/	0.01	1.43	1.45	1.41	1.84
//1	0.01	1.41	1.41	1.26	1.84
//2	0.00	1.47	1.45	1.57	1.90
//*	0.01	1.44	1.43	1.42	1.87

Table 2: Benchmarking of the subsequence generation algorithm against classic column generation. Settings are read as Maximum-candidate-set-size / Subsequence-identification-scheme / Pairing-enumeration-scheme. An asterisk means that an average is taken over that setting.

One should note that the number of uncovered flights have a very large impact on both the 'Root LP value ratio' and the 'Uncovered ratio'. If, for instance, one setting results in two out of 400 flights to be uncovered, and a second setting leaves three out of 400 flights uncovered, then the second setting would have a 'Uncovered ratio' of 1.5 when comparing to the first setting. The same holds for the 'Root LP value ratio', due to the high and dominant penalty for violating the flight constraints.

Lastly, from Table 2, it can be seen, that the subsequence generation algorithm has better integer measures on average. Therefore, we have very good reason to believe that integer solution can be found faster than when classic column generation is used.

Comparing the different settings of the subsequence generation algorithm, Table 2 shows, as expected, that the more the candidate subsequence set $\mathcal{C}$ is limited, the worse the solution quality gets, but the solutions also get slightly more integral, which is also expected. The different measures to identify attractive subsequences seem to perform almost equally good. This is probably due to a high correlation between them. Enumerating pairings over $\mathcal{C}$ gives a better solution quality than enumerating pairings over $\mathcal{L}$, but the integrality of the solution is higher when enumerating over $\mathcal{L}$. This is also expected, as $\mathcal{L}$ is more restricted than $\mathcal{C}$. Again, improving the subsequence identification would lead to increased solution quality, while integrality benefits of the restrictions could be kept.

Table 3 shows statistics for the test runs on the w08r03b instance. The statistics are representative for the other instances as well. The table reveals that almost no computation time is spent with analysis of columns and subsequence identification. As these are the two core parts of the solution approach, this clearly points out an area for future research with a great potential gain. If the right subsequences are identified, the solution quality would naturally increase.

Figure 7 shows a typical graph of the LP objective value per main loop iteration. The flat lines of the graph are especially interesting. Whenever there is a flat line, it means that the subsequences added in those iterations did not lower the LP objective value. This could mean one of two things: Either we have identified the wrong subsequence, or we have identified the right subsequence, but the subsequence cannot be used, so we do not get the gain of adding it. The latter is most easily seen in the case with unique subsequence for all flights. Let (f_1, f_2) be the subsequence selected in the current solution and let (f_1, f_3) be the new subsequence that is identified as attractive in the current iteration. However, (f_1, f_3) cannot be selected by the LP relaxation, before a new subsequence for (f_4, f_2) for f_2 is added. The dual vector will point out a suggestion for (f_4, f_2) eventually, but an early "subsequence partner"-prediction of (f_4, f_2) would be beneficial. The problem is, though, that this problem propagates much further than to just one other subsequence. Still, both cases support that an improvement of

Instance	Settings	Solve time (s)	LP solver (%)	Col. generation (%)	Col. analysis (%)	Subs. identification (%)	Col. enumeration (%)	Root LP value	Uncovered flights	“At 1” (%)	“Nice fraction” (%)	Main loop iterations	Subsequences added	Columns added	Stop reason
w08r03b	CG	3604.75	0	100	0	0	0	2.00034e+08	2	26	33	249	0	19811	TO
w08r03b	2/1/1	11.42	25	27	1	0	47	2.18032e+08	2	100	100	164	163	6298	NI
w08r03b	3/1/1	29.45	53	14	1	0	32	2.08365e+08	2	29	99	144	143	31669	NI
w08r03b	4/1/1	91.58	49	7	0	0	43	2.05233e+08	3	18	67	174	173	99385	NI
w08r03b	2/1/2	12.32	15	26	2	0	57	2.18698e+08	2	50	99	196	195	4075	NI
w08r03b	3/1/2	24.22	38	22	2	0	38	2.10430e+08	2	61	97	194	193	13904	NI
w08r03b	4/1/2	22.91	46	22	1	0	31	2.09529e+08	2	36	100	129	128	16949	NI
w08r03b	2/2/1	10.72	25	25	2	0	48	2.17366e+08	3	55	99	163	162	6590	NI
w08r03b	3/2/1	28.23	53	14	1	0	32	2.08366e+08	2	43	99	142	141	26443	NI
w08r03b	4/2/1	74.89	54	7	0	0	39	2.05306e+08	2	19	25	136	135	92115	NI
w08r03b	2/2/2	12.42	16	26	2	0	56	2.18032e+08	4	42	100	196	195	4374	NI
w08r03b	3/2/2	14.53	45	22	1	0	32	2.12528e+08	3	37	99	112	111	12596	NI
w08r03b	4/2/2	23.42	53	19	1	0	27	2.10027e+08	2	69	99	117	116	27339	NI
w08r03b	2/3/1	10.61	25	26	2	0	47	2.17432e+08	3	49	99	167	166	6356	NI
w08r03b	3/3/1	27.17	52	14	1	0	33	2.08366e+08	2	22	58	141	140	29918	NI
w08r03b	4/3/1	105.70	64	5	0	0	31	2.05233e+08	3	17	90	146	145	153388	NI
w08r03b	2/3/2	8.25	18	27	2	0	53	2.20247e+08	2	28	79	138	137	3606	NI
w08r03b	3/3/2	23.74	43	21	1	0	35	2.10363e+08	2	38	67	174	173	17832	NI
w08r03b	4/3/2	23.70	55	19	1	0	25	2.09630e+08	3	19	72	109	108	20837	NI
w08r03b	2/4/1	8.84	24	27	2	0	47	2.18367e+08	3	23	61	144	143	4679	NI
w08r03b	3/4/1	23.95	49	16	1	0	34	2.08365e+08	2	13	65	135	134	22741	NI
w08r03b	4/4/1	59.02	53	8	0	0	38	2.05533e+08	2	21	89	125	124	71852	NI
w08r03b	2/4/2	7.10	16	28	3	0	53	2.22532e+08	2	22	72	120	119	2840	NI
w08r03b	3/4/2	11.26	36	26	1	0	37	2.14027e+08	4	54	99	105	104	8726	NI
w08r03b	4/4/2	19.76	51	21	1	0	26	2.08531e+08	2	49	99	107	106	16682	NI
w08r03b	2/5/1	8.21	23	27	1	0	49	2.18032e+08	2	54	99	135	134	5060	NI
w08r03b	3/5/1	21.49	47	15	1	0	36	2.09363e+08	2	27	66	123	122	23275	NI
w08r03b	4/5/1	64.11	46	8	0	0	46	2.05233e+08	3	12	37	127	126	85375	NI
w08r03b	2/5/2	11.20	15	26	3	0	55	2.18586e+08	4	20	25	177	176	3675	NI
w08r03b	3/5/2	12.15	36	24	1	0	38	2.12027e+08	3	51	98	109	108	12143	NI
w08r03b	4/5/2	23.85	53	19	1	0	27	2.09529e+08	2	58	99	117	116	20007	NI

Table 3: Test statistics for the subsequence generation algorithm and classic column generation. Settings are read as Maximum-candidate-set-size / Subsequence-identification-scheme / Pairing-enumeration-scheme. NI abbreviates that no improvement is found over the given iteration span, and TO abbreviates time-out.

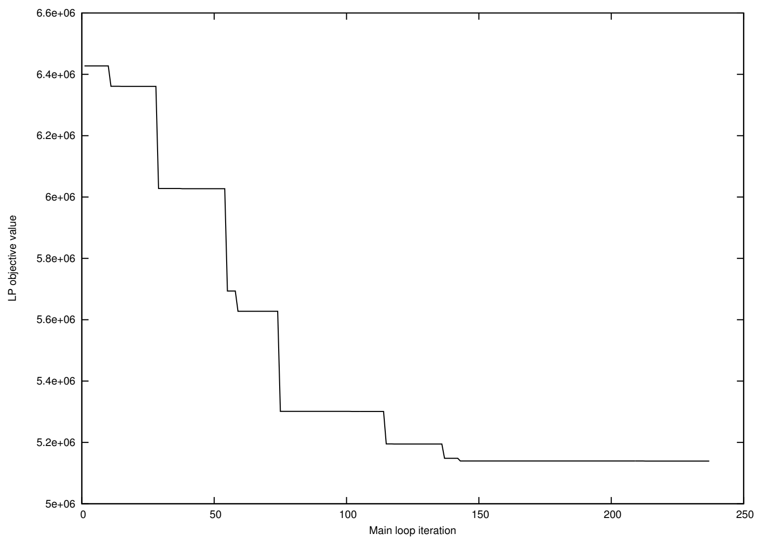


Figure 7: The LP objective value per main loop iteration for the instance w08r03b with settings 3/5/2.

the subsequence identification would benefit the overall algorithm a lot.

6 Conclusion and future work

We have contributed a novel solution approach based on generation of subsequences of flights for solving the well-known airline crew pairing problem. We have developed a method for solving the LP relaxation of the crew pairing problem, and the method aims at keeping the LP solution as close to integral as possible, thereby providing a good starting point for branch-and-bound. We have benchmarked the new method against a classic column generation algorithm. The benchmarking has revealed that the subsequence generation approach indeed is less fractional, but this comes at the price of a decrease in solution quality. The benchmarking has been carried out on real-life instances, and on all instances the subsequence generation approach was clearly faster than classic column generation.

There is a number of directions that future work on the subsequence generation method could go. Obviously, nesting the method in a branch-and-bound framework in order to find integer solutions would be interesting. This could be extended with delayed subsequence generation in each node of the branching tree. It should be noted that generation of subsequences in all tree nodes would mean that the objective value of a particular node is not a lower bound on the solution value for the children nodes.

The subsequence identification step is very important for the method to be successful. As mentioned earlier, the gain from adding a subsequence might not appear before another subsequence is added. Therefore, some kind of "subsequence partner"-prediction could prove beneficial. Perhaps also new measures for subsequence identification could be invented to complement the existing four measures. Still, as mentioned earlier, such measures should be computationally fast in order to avoid a large overhead of the approach. It could also be interesting to experiment with the outcome of identifying more than a single subsequence per iteration.

To use of historic data could also be a key to success. If one has a set of pairings that make up a good solution for June, then many of the subsequences in these pairings would probably be repeated in a good solution for July, as flight timetables and crew resources are relatively stable. Therefore, last month's subsequences could be put in the initial limited subsequence set, or a measure that in some way took these into account could be used.

As it is now the limited subsequence set $\mathcal{L}$ can only expand. It could speed up computation and maybe improve integrality, if the set could also shrink. Subsequences that have not appeared, i.e. columns containing the subsequence have not been selected even fractionally, in the LP solution for a given span of iterations could be removed. When a subsequence is removed, all columns containing the subsequence should be removed.

At the moment the candidate subsequence set $\mathcal{C}$ is static. However, there might be a gain in making it dynamic, as is the case for the limited subsequence set. Regarding the column enumeration, it is possible to generate duplicate columns. These could also be removed from the problem, if the overhead for doing this is not to severe.

A final suggestion for future work is to let pairing generation happen in parallel. Each of the $\mathcal{C}^k$ candidate subsequence subsets could be run on its own processor, and then return negative reduced cost columns to a single controlling process. As the pairing generation is faster than solving the LP relaxation, subsequences could be enumerated and added to the LP relaxation, before the LP solver had reached its optimum. Dual stabilisation should then be added in order to make the duals reliable as early as possible.

Acknowledgements: The authors would like to thank Paul Keating from Air New Zealand for providing the data instances and explaining them thoroughly, as well as his participation in many stimulating discussions about the solution approach.

References

S. AhmadBeygi, A. Cohn, and M. Weir. An integer programming approach to generating airline crew pairings. *Computers & Operations Research*, 36 (4):1284–1298, 2009.

E. Andersson, E. Housos, N. Kohl, and D. Wedelin. Crew pairing optimization. In G. Yu, editor, *Operations Research in the Airline Industry*, chapter 8, pages 228–258. Kluwer Academic Publishers, 1998.

C. Barnhart, L. Hatay, and E. L. Johnson. Deadhead selection for the long-haul crew pairing problem. *Operations Research*, 43(3):491–499, 1995.

C. Barnhart, A. M. Cohn, E. J. Johnson, D. Klabjan, G. L. Nemhauser, and P. H. Vance. Airline crew scheduling. In R. W. Hall, editor, *Handbook of Transportation Science*, chapter 14, pages 517–560. Kluwer Academic Publishers, Norwell, 2 edition, 2003.

E. R. Butchers, P. R. Day, A. P. Goldie, S. Miller, J. A. Meyer, D. M. Ryan, A. C. Scott, and C. A. Wallace. Optimized crew scheduling at air new zealand. *Interfaces*, 31(1):30–56, 2001.

M. Conforti, G. Cornuéjols, A. Kapoor, and K. Vuskovic. Perfect, ideal and balanced matrices. *European Journal of Operational Research*, 133 (3):455–461, 2001.

G. Desaulniers, J. Desrosiers, M. Gamache, and F. Soumis. Crew scheduling in air transportation. In T. G. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, pages 169–185. Kluwer Academic Publishers, Boston, 1998.

M. Ehrgott and D. M. Ryan. Constructing robust crew schedules with bicriteria optimization. *Journal of Multi-Criteria Decision Analysis*, 11 (3):139–150, 2002.

B. Gopalakrishnan and E. L. Johnson. Airline crew scheduling: State-of-the-art. *Annals of Operations Research*, 140(1):305–337, 2005.

G. W. Graves, R. D. McBride, I. Gershkoff, D. Anderson, and D. Mahidhara. Flight crew scheduling. *Management Science*, 39(6):736–745, 1993.

S. Irnich and G. Desaulniers. Shortest path problems with resource constraints. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation*, GERAD 25th Anniversary Series, chapter 2, pages 33–65. Springer, 2005.

S. Lavoie, M. Minoux, and E. Odier. A new approach for crew pairing problems by column generation with an application to air transportation. *European Journal of Operational Research*, 35(1):45–58, 1988.

D. M. Ryan and J. C. Falkner. On the integer properties of scheduling set partitioning models. *European Journal of Operational Research*, 35(3): 442–456, 1988.

M. Saddoune, G. Desaulniers, I. Elhallaoui, and F. Soumis. Integrated airline crew scheduling: A bi-dynamic constraint aggregation method using neighborhoods. *European Journal of Operational Research*, 212(3):445–454, 2011.

P. H. Vance, C. Barnhart, E. L. Johnson, and G. L. Nemhauser. Airline crew scheduling: A new formulation and decomposition algorithm. *Operations Research*, 45(2):188–200, 1997.

D. Wedelin. An algorithm for large scale 0-1 integer programming with application to airline crew scheduling. *Annals of Operations Research*, 57: 283–301, 1995.

Good and fast solutions to the airline crew pairing problem are highly interesting for the airline industry. In the solution method of this work we severely limit the number of allowed subsequent flights. Set partitioning problems with limited subsequences are known to be easier to solve, resulting in a decrease in solution time. The problem though is that a small number of deep subsequences might be needed for an optimal or near-optimal solution and these might not have been included by the subsequence limitation. Therefore, we try to identify or generate such subsequences that potentially can improve the solution value.

DTU Management Engineering
Department of Management Engineering
Technical University of Denmark

Produktionstorvet
Building 424
DK-2800 Kongens Lyngby
Denmark
Tel. +45 45 25 48 00
Fax +45 45 93 34 35

www.man.dtu.dk