A Subsequence Generation Approach for the Airline Crew Pairing Problem

Matias Sevel Rasmussen*1, Richard M. Lusby1, David M. Ryan2, and Jesper Larsen1

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

May, 2011

Abstract

In this paper we consider an important problem for the airline industry. The widely studied crew pairing problem is typically formulated as a set partitioning problem and solved using the branch-and-price methodology. Here we develop a new integer programming framework, based on the concept of subsequence generation, for solving the set partitioning formulation. In subsequence generation one restricts the number of permitted subsequent flights, that a crew member can turn to after completing any particular flight. By restricting the number of subsequences, the number of pairings in the problem decreases. The aim is then to dynamically add attractive subsequences to the problem, thereby increasing the number of possible pairings and improving the solution quality. Encouraging results are obtained on 19 real-life instances supplied by Air New Zealand and show that the described methodology is a viable alternative to column generation.

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

*Corresponding author: E-mail: mase@man.dtu.dk. Address: Department of Management Engineering, Technical University of Denmark, Produktionstorvet, Building 424, DK-2800 Kgs. Lyngby, Denmark. Tel.: +45-45254442. Fax: +45-45933435.
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 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 appear 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 pairings containing that subsequence is enumerated and added to the LP relaxation. Integrality is enforced using follow-on branching. 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 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 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 describe how integer requirements are enforced. In Section 6, 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 7, 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 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 $A$, 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

$$\begin{align*}
\text{minimise} & \quad c^\top x \\
\text{subject to} & \quad Ax = 1 \\
& \quad x \in \{0, 1\}^n.
\end{align*}$$

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 originates 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 $A$ to an $m \times n$ matrix where

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

Figure 2: Illustration of a pairing.
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:

\[
\begin{align*}
\text{minimise} & \quad c^\top x \\
\text{subject to} & \quad Ax = b \\
& \quad x \in \{0, 1\}^n.
\end{align*}
\]

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.

### 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 \( S(f) \subset \mathcal{F}^2 \). Subsequences are illustrated on Figure 3(a). In general terms for an \( m \times n \) zero-one matrix \( A \) with entries \( a_{ij} \), the subsequence set \( S(s) \), for any row \( s \) is given by

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

In the example on Figure 4 we have \( S(1) = \{(1, 3), (1, 4), (1, 6)\} \). Matrices where the subsequence count \( |S(s)| \leq 1 \) 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 balanced \( A \) 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
Figure 3: Subsequences for an ingoing flight.

Figure 4: Subsequences for row 1.
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 $L(f) \subseteq S(f)$. Let $S = \bigcup_{f \in F} S(f)$ denote the set of all subsequences for all flights, let $L = \bigcup_{f \in F} L(f)$ denote the set of limited subsequences for all flights, and let $O$ denote an optimal subsequence set, i.e. an optimal solution, for all flights. Naturally, $O$ 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 $L$ 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 $C(f)$ is defined for all flights $f \in F$, and again we define $C = \bigcup_{f \in F} C(f)$. We have $L(f) \subseteq C(f) \subseteq S(f)$ for all $f \in F$ and $L \subseteq C \subseteq S$, which is shown in Figure 5(b). The idea is now to expand $L$ with attractive subsequences from $C$. A subsequence $s \in C$ is attractive if it is likely that $s \in O$, where again $O$ is a set of optimal subsequences. Iteratively an attractive subsequence set $A \subseteq C$ is found and added to $L$. Although Figure 5(b) shows the set of optimal subsequences to be contained in the candidate subsequence set, there is no guarantee for this. In Algorithm 1, the outline of the algorithm for solving the LP relaxation of the pairing problem can be seen.

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 $C$. The pairing generator is a resource constrained shortest path solver, which is run on subsequences from $C$. The shortest path solver is a labelling algorithm, see for instance Irnich and Desaulniers (2005).
(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 $\mathcal{O}$.

**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
negative reduced cost columns, that are returned from the pairing generator, are analysed in order to collect statistics about the subsequences in $C \setminus L$.

The pairing generator is run sequentially on $N$ different networks consisting of subsequences $C^k \subseteq C$ for $k \in \{1, \ldots, N\}$ with $\bigcup_{k=1}^{N} C^k = C$. 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)](Ehrgott) 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 $C$ (so we have $N = 3$), based on these classes. The set $C^1$ consists of follow-the-aircraft subsequences and robust subsequences. The set $C^2$ consists of follow-the-aircraft subsequences and meal break subsequences, and $C^3$ 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 $|C^k(f)| \leq n^k$, where $n^k$ is a small integer less than, say, five for all $k \in \{1, \ldots, N\}$. 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 $C^1$ is used as the initial limited subsequence set $L$, where total enumeration is carried out.

For each subsequence $s \in C$ 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 $A$. 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 $A$. 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.

At each iteration some subsequences are identified as attractive based on these four measures and added to the set $\mathcal{L}$. Once a subsequence $s$ has been identified as an attractive subsequence, an enumeration procedure is performed to generate a whole set of new pairings, which all contain the identified subsequence. All enumerated pairings are then added to the relaxed master problem. The enumeration returns feasible pairings in $\mathcal{L}$ containing $s$. The reason why a relatively large set of columns is added to the LP model, is, that whenever a subsequence is identified as attractive, it is believed that it is likely to end up in an optimal solution. That is, the optimal solution should contain one of the enumerated columns. With this enumeration scheme, adding a subsequence only increases the subsequence count with one for a single flight, namely 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.
5 Integer programming framework

Given the preceding section on subsequence generation, one can obtain a high quality solution to the relaxed master problem. Since only a subset of the subsequences are considered this approach cannot provide a certificate of optimality. In order to solve the airline crew pairing problem we need a solution that satisfies the integral restrictions of Model (1)–(3). In this section we present an integer programming framework, which utilises follow-on branching, to force the $x_j$ variables to assume integer values.

5.1 Solving the LP relaxation

Figure 6(b) provides a schematic view of how the subsequence generation procedure solves an instance of the relaxed master problem. To initialise the algorithm all possible pairings from the subsequence set $C^1$ are enumerated and given to the LP solver to obtain an initial solution. The dual solution to this problem is then passed to the subsequence generation routine, which executes a series of pairing generators. Each pairing generator returns a set of negative reduced cost pairings which are then analysed in order to identify one or more attractive subsequences. Upon identifying such a subsequence an enumeration procedure is performed to generate all pairings that contain the specified subsequence. The relaxed master problem is then re-optimised with additional set of pairings. One iterates in this fashion until one of the following situations occurs:

1. No significant improvement in the objective function for a specified number of iterations.
2. No negative reduced cost pairings are returned from the pairing generators.

It can be observed that the process is quite similar to column generation. However, with column generation, the dual solution is passed to the pairing generators and any negative reduced cost pairings are added directly to the relaxed master problem.

5.2 Follow-on branching

Fractional solutions to the relaxed master problem arise when two or more pairings compete to cover the same flight(s). Due to the set partitioning structure of the model, one knows that in any optimal solution at most one pairing can cover any flight. In most airline crew pairing applications the branching method of choice is the so-called follow-on branching rule. This rule is a variation of the constraint branching technique developed by Ryan and Foster (1981) and is also what is implemented in this paper. In
follow-on branching one must identify two flights that are flown consecu-
tively and contained in a pairing that is covered fractionally (i.e. at a value
greater than zero, but strictly less than one) in a solution to the relaxed
master problem. This branching strategy partitions the solution space into
two disjoint subspaces (or branches). The first ensures that the two flights
are flown consecutively, while the second ensures that they are covered by
different pairings. Since one is branching on consecutive flights, this rule is
particularly easy to incorporate in the pairing generators as it only requires
the modification of arcs associated with two flights in the network. Further-
more, the follow-on branching concept is closely related to the notion of a
subsequence—identifying two flights to be flown consecutively amounts to
identifying a subsequence.

To identify the subsequence to branch on given a fractional solution
to a relaxed master problem we simply find the subsequence that is covered
fractionally at maximum value (i.e. at a value greater than zero, but strictly
less than one) and create two new nodes to be solved, as outlined above.
Imposition of a branch requires one to first remove those pairings that violate
the branch from the relaxed master problem. Here, we simply bound all such
variables to zero. As we mentioned in Section 2 we allow the partitioning
flight constraints and base constraints to be violated, with an appropriate
penalty. The artificial variables are never bounded to zero and ensure we
always have a starting basis after this bounding step. By retaining the
artificial variables, we do not need to implement a time-consuming phase
1/phase 2 approach.

Through modifications to the pairing generators, any pairing that vi-
olates the branch is prevented from entering the problem. If the branch
states that the subsequence should not be contained in any pairing (i.e. the
two flights cannot be flown consecutively), then the corresponding arc is
removed from any pairing generators it appears in. If, on the other hand,
one is forcing a subsequence to be contained in a pairing, then one must
ensure that the corresponding arc is contained in the solutions to the re-
spective resource constrained shortest paths. Here, to enforce a particular
subsequence, we remove all other conflicting subsequences from the relevant
networks. This ensures that the desired subsequence is contained and pre-
vents us from having to introduce a new resource in the shortest path solve.
A conflicting subsequence is one in which either the inbound or outbound
flight is different to that stated in the subsequence to branch on. Figure 7
illustrates how a subsequence (consisting of flights one and two) is enforced.
The arcs given in red are all subsequences that must be removed in order to
ensure that flight 1 and flight 2 are flown consecutively (or not flown at all)
in a pairing.
5.3 Solving the integer programme

To produce a high quality solution to Model (1)–(3), we combine the follow-on branching strategy of the preceding section with the subsequence generation methodology of Section 5.1 to implement a kind of branch-and-price algorithm. Branch-and-price is a well-known technique, which utilises column generation, for solving the crew pairing problem (see Barnhart et al. (1998) for details). Due to the fact that we identify, and add dynamically, new subsequences to the problem as we proceed, even during the branching phase of the approach, the integer programming framework we propose does not strictly adhere to traditional branch-and-bound principles. In particular, one cannot guarantee that a child node will have an objective function value that is at least that of its parent. We are prepared to make this sacrifice, since as it is hoped that by identifying good subsequences at the root node high quality integer solutions can be obtained quickly (without too much branching).

When solving nodes of the branch-and-bound tree we adopt a depth-first strategy, each time enforcing the identified subsequence, since this often produces a good integer solution quickly. Upon finding the first integer solution, however, we switch to a best-first search. That is, we evaluate the unexplored nodes in increasing order of their parent’s objective value. Figure 6(b) with two modifications, can be considered the solution procedure for any node. When initialising the algorithm all pairings that do not satisfy the branch to enforce must be removed. Furthermore, all networks must be modified to ensure only feasible pairings (i.e. all necessary branches are enforced) are generated. The branch-and-bound procedure terminates when all nodes have been evaluated or the incumbent integer solution is within a degree of tolerance of the best, unexplored node. While this integer programming approach does not provide valid lower bounds, the idea of subsequence generation is to provide a good integer solution quickly. In the computational results of Section 6 we compare our integer solutions to the optimal solution of the relaxed master problem.
Table 1: Characteristics for the test instances.

<table>
<thead>
<tr>
<th>Instance</th>
<th>F</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>w08r01a</td>
<td>450</td>
<td>5</td>
</tr>
<tr>
<td>w08r01b</td>
<td>430</td>
<td>5</td>
</tr>
<tr>
<td>w08r01c</td>
<td>370</td>
<td>5</td>
</tr>
<tr>
<td>w08r01d</td>
<td>400</td>
<td>4</td>
</tr>
<tr>
<td>w08r01e</td>
<td>380</td>
<td>5</td>
</tr>
<tr>
<td>w08r02a</td>
<td>350</td>
<td>5</td>
</tr>
<tr>
<td>w08r02b</td>
<td>400</td>
<td>4</td>
</tr>
<tr>
<td>w08r02c</td>
<td>320</td>
<td>5</td>
</tr>
<tr>
<td>w08r02d</td>
<td>450</td>
<td>5</td>
</tr>
<tr>
<td>w08r02e</td>
<td>350</td>
<td>4</td>
</tr>
<tr>
<td>w08r03a</td>
<td>320</td>
<td>4</td>
</tr>
<tr>
<td>w08r03b</td>
<td>420</td>
<td>5</td>
</tr>
<tr>
<td>w08r03c</td>
<td>420</td>
<td>5</td>
</tr>
<tr>
<td>w08r03d</td>
<td>450</td>
<td>5</td>
</tr>
<tr>
<td>w08r03e</td>
<td>320</td>
<td>4</td>
</tr>
<tr>
<td>w08r04a</td>
<td>400</td>
<td>4</td>
</tr>
<tr>
<td>w08r04b</td>
<td>400</td>
<td>5</td>
</tr>
<tr>
<td>w08r04c</td>
<td>350</td>
<td>5</td>
</tr>
<tr>
<td>w08r04d</td>
<td>320</td>
<td>5</td>
</tr>
<tr>
<td>w08r04e</td>
<td>400</td>
<td>4</td>
</tr>
<tr>
<td>w08r05a</td>
<td>330</td>
<td>5</td>
</tr>
</tbody>
</table>

6 Computational results

In this section we analyse the performance of the proposed solution approach on 19 real-life data instances that were made available to us by Air New Zealand. The data sets are taken from Air New Zealand’s domestic timetable. The does, however, also include destinations in Australia and the Pacific Islands. To perform the computational analysis we restrict the number of flights in each of the instances. This is done to ensure that they terminate in reasonable time. Table 1 states the number of flights ($|F|$) and the total number of base constraints for each instance ($|B|$).

There are a number of parameters one must determine when implementing the subsequence generation. These include the following:

1. Which criterion does one use to identify a subsequence to add to the problem?

2. How many subsequences should at most be contained in each of the sets $C^1$, $C^2$, and $C^3$?

Based on the findings from Rasmussen et al. (2011), the selection strategy that is used to identify an entering subsequence is a round-robin procedure which loops over the four measures described in Section 4. At each iteration the subsequence which has the highest score in the measure under consideration is added to the problem. Enumeration is then performed. Here, we test and compare the impact on solution time and quality by increasing the number of subsequences that can be contained in the sets $C^1$, $C^2$, and $C^3$. In the first case we restrict the sets to contain at most three subsequences, while in the second this limit is set to four. The branching routine terminates when the incumbent integer solution is within 1% of the “best” unexplored node. We impose a time limit of 3600 seconds on the complete algorithm. The penalties associated with not covering a flight and violating a base constraint are $10^8$ and $10^6$, respectively. 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 6.0 using an academic license.

Table 2 gives the results for the case in which each of the subsequence classes $C^1$, $C^2$, and $C^3$ contains at most three subsequences, i.e. $n^k = 3$. For
<table>
<thead>
<tr>
<th>instance</th>
<th>Best Integer</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>obj time (s)</td>
<td>root LP*</td>
</tr>
<tr>
<td>w08r01a</td>
<td>6.12041e+08</td>
<td>323.53</td>
</tr>
<tr>
<td>w08r01b</td>
<td>3.07034e+08</td>
<td>246.09</td>
</tr>
<tr>
<td>w08r01c</td>
<td>3.05040e+08</td>
<td>634.48</td>
</tr>
<tr>
<td>w08r01d</td>
<td>2.10030e+08</td>
<td>117.34</td>
</tr>
<tr>
<td>w08r01e</td>
<td>8.21038e+08</td>
<td>280.26</td>
</tr>
<tr>
<td>w08r02a</td>
<td>5.00023e+08</td>
<td>19.67</td>
</tr>
<tr>
<td>w08r02b</td>
<td>3.00023e+08</td>
<td>249.96</td>
</tr>
<tr>
<td>w08r02c</td>
<td>8.00019e+08</td>
<td>202.31</td>
</tr>
<tr>
<td>w08r02d</td>
<td>3.00029e+08</td>
<td>468.04</td>
</tr>
<tr>
<td>w08r02e</td>
<td>5.18034e+08</td>
<td>281.61</td>
</tr>
<tr>
<td>w08r03a</td>
<td>8.01024e+08</td>
<td>241.71</td>
</tr>
<tr>
<td>w08r03b</td>
<td>2.11028e+08</td>
<td>184.79</td>
</tr>
<tr>
<td>w08r03c</td>
<td>2.12030e+08</td>
<td>360.95</td>
</tr>
<tr>
<td>w08r03d</td>
<td>3.12030e+08</td>
<td>455.32</td>
</tr>
<tr>
<td>w08r03e</td>
<td>8.19025e+08</td>
<td>102.07</td>
</tr>
<tr>
<td>w08r04a</td>
<td>4.09027e+08</td>
<td>12.69</td>
</tr>
<tr>
<td>w08r04b</td>
<td>4.17025e+08</td>
<td>345.34</td>
</tr>
<tr>
<td>w08r04c</td>
<td>3.16025e+08</td>
<td>8.22</td>
</tr>
<tr>
<td>w08r04d</td>
<td>8.09021e+08</td>
<td>16.29</td>
</tr>
</tbody>
</table>

Table 2: Classes $C^1$, $C^2$, and $C^3$ have at most three subsequences.

Each instance we state the instance name, the objective value of the best integer solution, and the time at which this solution was obtained. Furthermore, we provide an indication of the quality of this solution through a comparison with objective function value of the relaxed master problem (root LP) as well as the objective function value of the relaxed master problem obtained using a conventional column generation procedure. The column generation procedure has no restrictions on the number of subsequences each flight can have and is also given a time limit of 3600 seconds. For a fair comparison, we also hot start this procedure with an enumeration of pairings on the $C^1$ subsequence class. On all instances the column generation procedure timed out and what is given in the table is the objective function value of the root node at termination (cg LP). We also provide the percentage gap between the objective value of our integer solution and the solution obtained using column generation, the number of uncovered flights in our solution (UF), the number of nodes explored in the branching phase of the algorithm, and the time required for the algorithm to execute. The subsequence generation procedure terminates when the incumbent integer solution is within 1% of the “best” unexplored. That is, within 1% of objective value of the best node’s parent.

One can observe from the table that the subsequence generation procedure provides, with a few exceptions, good quality integer solutions (within a few percent of the objective value obtained using column generation) given the one hour time limit. In all cases an integer solution is obtained within 11 minutes of computation time. The column generation procedure, on the other hand, does not converge within the same time frame. It is encouraging.
to see that for some instances (i.e. w08r02a and w08r04a) we are within 2% of the column generation approach extremely quickly. However, instances w08r01e, w08r02c, w08r03a, w08r03e, and w08r04d show that there is room for improvement in the subsequence identification phase of the algorithm. The large percentage gaps can be explained by the fact that we have more uncovered flights than the column generation procedure. For example, for instance w08r03a we have twice as many uncovered flights. However, to put this in perspective, w08r03a is a flight schedule containing 320 flights and we uncover eight of them. Comparing the time at which the best integer solution was found with the time it took the algorithm to conclude, we note that all instances terminated upon finding the first integer solution. One can also see that in several cases the integer solution obtained has a better objective function value than the root node. As we mentioned in Section 5.3, this is possible as subsequences are dynamically added during the branching phase of the algorithm.

Table 3 gives the results for the case in which each of the subsequence classes $C^1$, $C^2$, and $C^3$ contains at most four subsequences, i.e. $n^k = 4$. While this table reinforces many of the conclusions from Table 2, one can also observe that the integer solutions are slightly better than those obtained in Table 2. Increasing the class sizes does, however, slow the method down. This can be explained by the following. A larger candidate set of subsequences creates larger networks for the pairing generators and in doing so creates more feasible pairings. As a result, the enumeration procedure not only takes longer, but there are also more pairings in the relaxed master problem making the optimization slower. Interestingly, instance w08r04d is the only instance for which we uncover fewer flights by increasing the candidate subsequence set size. This could be a result of increased flexibility given the additional flights or a result of the subsequence generation taking a different path in the execution of the algorithm. That is, subsequences are identified in a different order, prompting a different sequence of events in the algorithm.

Finally, the fact that in some cases we uncover more flights than would appear necessary would suggest that a more sophisticated process of including subsequences in the candidate set might be required. Figure 8 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...
<table>
<thead>
<tr>
<th>instance</th>
<th>Best Integer</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>obj</td>
<td>time (s)</td>
</tr>
<tr>
<td>w08r01a</td>
<td>6.10039e+08</td>
<td>234.22</td>
</tr>
<tr>
<td>w08r01b</td>
<td>3.04036e+08</td>
<td>75.48</td>
</tr>
<tr>
<td>w08r01c</td>
<td>3.02037e+08</td>
<td>1107.07</td>
</tr>
<tr>
<td>w08r01d</td>
<td>2.09031e+08</td>
<td>1371.26</td>
</tr>
<tr>
<td>w08r01e</td>
<td>8.19041e+08</td>
<td>295.70</td>
</tr>
<tr>
<td>w08r02a</td>
<td>5.00022e+08</td>
<td>59.37</td>
</tr>
<tr>
<td>w08r02b</td>
<td>3.00023e+08</td>
<td>307.40</td>
</tr>
<tr>
<td>w08r02c</td>
<td>8.00018e+08</td>
<td>48.34</td>
</tr>
<tr>
<td>w08r02d</td>
<td>3.00026e+08</td>
<td>1207.20</td>
</tr>
<tr>
<td>w08r02e</td>
<td>5.17030e+08</td>
<td>699.73</td>
</tr>
<tr>
<td>w08r03a</td>
<td>8.00021e+08</td>
<td>14.55</td>
</tr>
<tr>
<td>w08r03b</td>
<td>2.06033e+08</td>
<td>1343.80</td>
</tr>
<tr>
<td>w08r03c</td>
<td>2.08033e+08</td>
<td>1080.62</td>
</tr>
<tr>
<td>w08r03d</td>
<td>3.11031e+08</td>
<td>19.50</td>
</tr>
<tr>
<td>w08r03e</td>
<td>8.15024e+08</td>
<td>574.02</td>
</tr>
<tr>
<td>w08r04a</td>
<td>4.07028e+08</td>
<td>1333.81</td>
</tr>
<tr>
<td>w08r04b</td>
<td>4.14028e+08</td>
<td>183.32</td>
</tr>
<tr>
<td>w08r04c</td>
<td>3.13027e+08</td>
<td>12.62</td>
</tr>
<tr>
<td>w08r04d</td>
<td>7.05023e+08</td>
<td>159.30</td>
</tr>
</tbody>
</table>

Table 3: Classes $C^1$, $C^2$, and $C^3$ have at most four subsequences.

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 the subsequence identification would benefit the overall algorithm a lot.

7 Conclusion and future work

In this paper we have described a new integer programming framework for solving the well-known airline crew pairing problem. At the core of the methodology is subsequence generation. This approach limits the number of subsequences in the problem and dynamically adds attractive subsequences as needed. A follow-on branching strategy is described for obtaining integer solutions. Encouraging results are presented for 19 real-life instances supplied by Air New Zealand. In comparison to a column generation procedure that fails to converge to the optimal solution of the relaxed master problem within an hour of computation time, the methodology presented in this paper produces good quality integer solutions well within the same time limit. This indicates the method could potentially be a viable alternative to the conventional column generation approach to this problem.

While the results are encouraging, they also suggest that improvements are necessary. For instance, as it is now, the candidate set of subsequences $\mathcal{C}$ is a static set. If this does not contain the optimal subsequences (or at least a close to optimal set), then it is unlikely the method will do well. One
promising improvement would be to be make this set dynamic so that one could add new candidate subsequences during the solution process, or even remove some unpromising ones. In this way one can keep a good, small set of subsequences.

Furthermore, one can also improve the subsequence identification step. This is the core process in the approach and dictates how many pairings will be added to the problem. Improvements here will positively impact the run time of the approach. 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.

A final suggestion for future work is to let pairing generation happen in parallel. Each of the $C^k$ 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**


