An IP Framework for the Crew Pairing Problem Using Subsequence Generation

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

Publication date:
2011

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

An IP Framework for the Crew Pairing Problem using Subsequence Generation

Report 10 2011

DTU Management Engineering

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

June 2011

DTU Management Engineering
Department of Management Engineering

An IP Framework for the Crew Pairing Problem using Subsequence Generation

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

June, 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, Subsequence generation, Set partitioning, Integer programming

1 Introduction

For an airline company crew costs can be identified as the second largest expense, typically only fuel costs are higher. In 1991 it was reported that

*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.

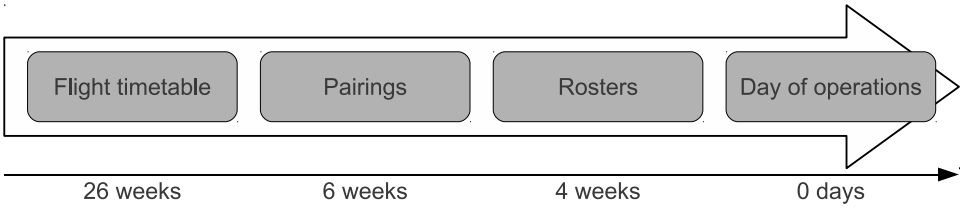

Figure 1: The airline crew scheduling process. The times are for Air New Zealand's domestic scheduling.

American Airlines spent USD 1.3 billion on crew (see Gopalakrishnan and Johnson (2005)). Unlike fuel costs, crew costs can in some sense be controlled by an airline. The inefficient use of crew may lead to unnecessary expenditure. As a result, finding the optimal use of an airline's crew is a topic that has received significant attention in the literature, and now, as a result, optimisation tools are heavily used by airlines in their planning operations. Due to the high cost associated with crews, even minor improvements in the schedules can result in significant savings.

The focus of this paper is on the so-called *airline crew pairing problem*. This problem is one of the core optimisation problems encountered by an airline company. It's position in the planning horizon, as well as the other main optimisation problems can be seen in Figure 1. While the airline crew pairing problem will be discussed in more detail in Section 2, it essentially requires one to find sequences of flights that crew members will fly, at minimum cost. The sequences of flights are termed *pairings* and are anonymous. The pairing problem succeeds the *flight timetabling* step. Here, a schedule of all flights that will be flown by the airline must be constructed. This is then followed by *fleet assignment*, which requires one to assign an aircraft type to each of the flights. Finally, as a last step from an aircraft perspective, one must determine aircraft routes, termed the *aircraft routing problem*. A solution to the flight timetabling phase is required as input for the pairing problem; however, one can solve the pairing problem separately for cockpit and cabin crew, or even separately for each aircraft type. For example, here we consider the domestic Boeing 737 fleet for Air New Zealand. The final step in the planning horizon is *crew rostering*. Here, pairings, training, and vacation are combined to form actual rosters for individual crew member. The crew pairing and the crew rostering steps are together called *airline crew scheduling* The steps for airline crew scheduling can be seen in Figure 1.

A recent survey on airline crew scheduling can be found in Gopalakrishnan and Johnson (2005). The authors provide an overview of the different approaches that have been used over the last two decades to solve this prob-

lem. In addition to this, some promising directions for future work are described. The crew pairing problem has also been 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. The crew pairing problem is formulated as a set partitioning problem and the authors describe how the problem can be solved as a weekly problem or a dated problem. In the weekly problem approach, one exploits repetitive patterns of flights over the weekdays, and can thus 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 optimisation approach 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. The crew pairing problem is solved by 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. The authors also formulate the crew pairing problem as a set partitioning problem. Lavoie et al. (1988) use a set covering formulation and perform column generation on a duty period network. 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 constraints belonging to the same pairing are just represented by one active constraint. The update of these active constraints leads 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.

In this paper we extend the work of Rasmussen et al. (2011), where the idea of using subsequence generation for solving the pairing problem was first proposed. Given the encouraging results from a linear programming (LP)

perspective, here we extend this methodology to a full integer programming framework. The framework itself is similar to that of the well known *branch-and-price* approach for solving large scale optimisation problems; however, it does possess certain key differences. In particular, pairings found during the pricing phase of the algorithm are not directly added to the master problem, but instead are analysed in order to identify attractive subsequences. This is then followed by an enumeration step in which all pairings containing the identified subsequence(s) are enumerated and simultaneously added to the master problem.

The paper is organized as follows. In Section 2 we provide a definition of the airline crew pairing problem and present the conventional generalised set partitioning formulation of it. Section 3 describes a subsequence generation based solution approach for solving the linear programming model. This methodology is built into an integer programming framework in Section 4 and computational results for the complete algorithm are presented in Section 5. The main conclusions and directions for future work are summarised in Section 6.

2 Problem formulation

In this section we formally define the airline crew pairing problem and present a mathematical formulation of it. We begin by introducing the required terminology and sets.

The flight schedule of an airline consists of a set of flights, $\mathcal{F}$. A *duty period* is a sequence of flights in $\mathcal{F}$ that can be flown by a crew member. Any duty period must adhere to 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. Consecutive duty periods are separated by a *rest period*, which must have a minimum duration. Finally, a so-called *sign-on* (*sign-off*) time, is imposed when starting (respectively, terminating) a duty period.

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* (where one manday is equivalent to the amount of work one person can produce in a day). 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 described in this paper, the pairing's idle time is used as the cost of a pairing. This simple measure ensures the crew utilization is maximized. An illustration of a pairing can be seen on Figure 2.

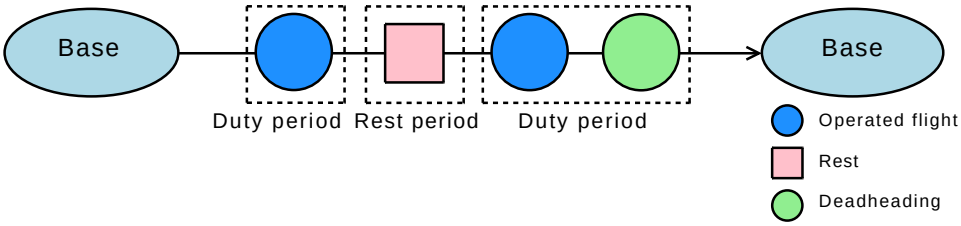

Figure 2: Illustration of a pairing.

The airline crew pairing problem entails finding a 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 $f \in \mathcal{F}$ and each column corresponds to a pairing $p \in \mathcal{P}$. 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 denote the cost of pairing $j \in \{1, \ldots, n\}$.

In addition to the flight partitioning constraints, most airlines include so-called *base constraints*. Such constraints are necessary in order to distribute the pairings across the crew bases in such a way that is consistent with where the actual crew are located. In other words, a base constraint is associated with a set of crew bases and puts a lower limit and/or an upper limit on the number of mandays that can be worked out of the associated bases over a given time horizon. A pairing contributes to a base constraint if the pairing originates, in the specified time horizon, from one of the bases associated with the constraint. The pairing's contribution to the base constraint is the manday count of the pairing and given as d_j, where $j \in \{1, \ldots, n\}$. Here we denote the set of base constraints that enforce an upper limit on manday count as $\mathcal{B}_1$, while those that enforce a lower limit are given by the set $\mathcal{B}_2$. The two matrices B_1 and B_2, with dimension $|\mathcal{B}_1| \times n$ and $|\mathcal{B}_2| \times n$, respectively, reflect the manday coverage of each pairing for each type of base constraint. Finally, vectors $\boldsymbol{b}_1 \in \mathbb{N}_0^{|\mathcal{B}_1|}$ and $\boldsymbol{b}_2 \in \mathbb{N}_0^{|\mathcal{B}_2|}$ give the corresponding upper and lower limits on manday count.

The decision variables x_j for $j \in \{1, \ldots, n\}$ govern the inclusion of pair-

ing j in the solution and are binary. We also allow the possibility for leaving flights uncovered. The decision variables s_i for $i \in \{1, \ldots, m\}$ give the possibility of leaving flight $i \in \mathcal{F}$ uncovered. Such variables are necessary to ensure feasibility and to reflect their unattractiveness each is assigned a high objective coefficient, M. In a similar way we also allow the base constraints to be violated. In reality one would prefer to violate the base constraints in preference to cancelling some of the flights. The decision variables u_k where $k = 1, \ldots, |\mathcal{B}_1|$ and o_k where $k = 1, \ldots, |\mathcal{B}_2|$ give the number of mandays one violates the respective base constraints by. Again, such decision variables are assigned a high cost, p (where $p < M$), to make them unattractive. The mathematical programme can then be written as follows:

$$
\begin{aligned}
\text{minimise} \quad & \boldsymbol{c}^\top \boldsymbol{x} + p\boldsymbol{s} + M\boldsymbol{u} + M\boldsymbol{o} && (1)\\
\text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} + \boldsymbol{I}\boldsymbol{s} = \boldsymbol{1} && (2)\\
& \boldsymbol{B}_1\boldsymbol{x} + \boldsymbol{I}_1\boldsymbol{u} = \boldsymbol{b}_1 && (3)\\
& \boldsymbol{B}_2\boldsymbol{x} - \boldsymbol{I}_2\boldsymbol{o} = \boldsymbol{b}_2 && (4)\\
& \boldsymbol{x} \in \{0,1\}^n && (5)\\
& \boldsymbol{s} \in \mathbb{R}^n && (6)\\
& \boldsymbol{u} \in \mathbb{R}_+^{|\mathcal{B}_1|} && (7)\\
& \boldsymbol{o} \in \mathbb{R}_+^{|\mathcal{B}_2|} \ . && (8)
\end{aligned}
$$

where $\boldsymbol{I}, \boldsymbol{I}_1$ and $\boldsymbol{I}_2$ are identity matrices of appropriate size.

The number of possible pairings in the set partitioning formulation is very large and the above model is typically only solved using branch-and-price methodology. In what follow, however, we will present a new integer programming framework for efficiently solving this problem based on the subsequence methodology described in Rasmussen et al. (2011).

3 Subsequence methodology

The idea of using subsequence generation to solve the airline crew pairing problem was first proposed in Rasmussen et al. (2011). The fundamental idea with this approach is to cleverly limit the number of subsequent flights that can feasibly follow any flight $f \in \mathcal{F}$ and then dynamically introduce attractive so-called *subsequences* during the solution process. Formally stated, the subsequences of a particular flight $f \in \mathcal{F}$ are the set of pairs $(f, g) \in \mathcal{F}^2$, where $g \in \mathcal{F}$ is a feasible subsequent flight of f. Figure 3 illustrates this concept. We denote the set of subsequences of any flight $f \in \mathcal{F}$ as $\mathcal{S}(f)$. The *subsequence count* of any flight $f \in \mathcal{F}$ is then given as $|\mathcal{S}(f)|$ and the term *unique subsequence* refers to the situation in which the subsequence count

of all flights $f \in \mathcal{F}$ is at most one. We denote the set of all subsequences as $\mathcal{S} = \bigcup_{f \in \mathcal{F}} \mathcal{S}(f)$, and $\mathcal{O}$ is used to refer to a set of optimal subsequences.

The premise is that by considering a limited subsequence set, one can reduce the complexity of the mathematical model and, if done in an intelligent manner, the approach should not reduce the quality of the solution. Due to the set partitioning constraints of the model, one can observe that an optimal integer solution must have unique subsequence. In what follows, we discuss how one first limits the number of subsequences and then how one identifies attractive subsequences to add to the problem.

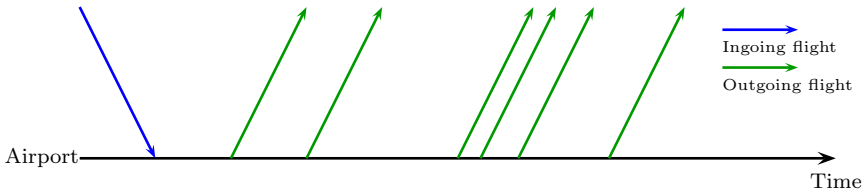

Figure 3: Subsequences.

3.1 Subsequence limitation

Limiting the number of possible subsequences for each flight in the problem is beneficial from a computational perspective. Naturally it reduces the total number of pairings that must be considered. In addition to this, however, from a graph theoretical perspective problems with a limited subsequence set produce more balanced constraint matrices. That is, the constraint matrix defines a polytope that has integral or near integral vertices (see Ryan and Falkner (1988) and Conforti et al. (2001) for details). Figure 4 provides an example of a limited subsequence set for an incoming flight $f \in \mathcal{F}$. The set $\mathcal{L}(f) \subseteq \mathcal{S}(f)$ is used to denote the limited subsequence set for flight $f \in \mathcal{F}$. The set $\mathcal{L} = \bigcup_{f \in \mathcal{F}} \mathcal{L}(f)$ contains all limited subsequences for all flights. In the example $|\mathcal{L}(f)| = 2$ and $|\mathcal{S}(f)| = 6$. Being able to

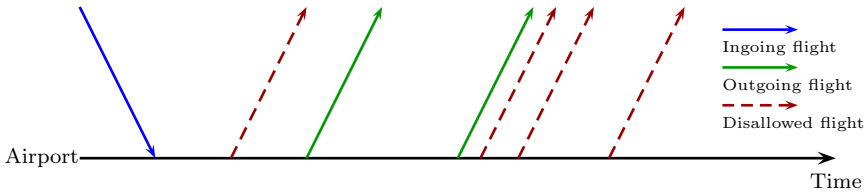

Figure 4: Limited subsequences.

accurately identify which subsequences to include in $\mathcal{L}(f)$ for every flight

$f \in \mathcal{F}$ is of crucial importance with this approach. Obviously, one must determine which subsequences to initially include. However, the limited subsequence set for each flight is a dynamic set in the sense that one can add attractive subsequences during the solution process. Here, attractive refers to a subsequence's ability to reduce the objective function of the *relaxed* master problem. The relaxed master problem is problem of the form of Model (1)–(8) that contains only a subset of all pairings and has no integral restrictions. The overall aim of the subsequence approach is to be able to identify a set of optimal subsequences without considering all possible subsequences in the pairing construction.

3.2 Subsequence generation

The purpose of subsequence generation is to use successive solutions to the relaxed master problem to identify an attractive subsequence (or possibly more than one) to add to the respective limited subsequence sets in order to enrich the set of possible pairings in the problem. Given the nature of the objective function it is unlikely that many *deep* subsequences will be in an optimal solution. A deep subsequence is one with a lengthy duration between the incoming and subsequent outbound flight. Therefore, we begin by defining *candidate* subsequence sets $\mathcal{C}(f)$ for each flight $f \in \mathcal{F}$. 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 $\mathcal{L} \subseteq \mathcal{C} \subseteq \mathcal{S}$. The relationship between these sets can be seen in Figure 5. Ideally we would like $\mathcal{O} \subset \mathcal{C}$. Subsequence generation is an iterative proce-

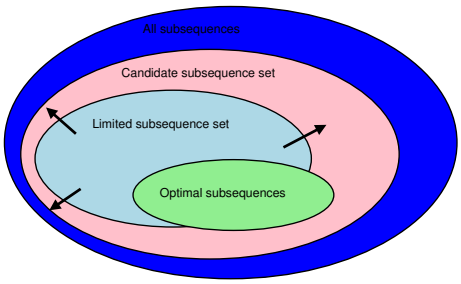

Figure 5: The relations between the set of all subsequences $\mathcal{S}$, the limited subsequence set $\mathcal{L}$, the candidate subsequence set $\mathcal{C}$, and an optimal subsequence set $\mathcal{O}$.

dure noticeably similar to column generation. At every iteration the relaxed master problem is solved and the dual vector is used to identify negative reduced cost pairings. Finding a negative reduced cost pairing entails solving a resource constrained shortest path problem on a connection network, see Clausen et al. (2010). Each node of this network corresponds to a particular flight $f \in \mathcal{F}$, while an arc between any two nodes indicates that it is possible

for one flight to follow the other feasibly in a pairing. Unlike column generation, however, the aim is not to return a pairing from the pairing generation step, but rather to identify a good subsequence that should be contained in a pairing. For this reason we set up a number of sparse networks specifically designed to identify particular subsequences. The following classes of subsequences are important to recognise in crew pairing:

1. Follow-the-aircraft
2. Robust
3. Meal break
4. Overnight

For any flight $f \in \mathcal{F}$, the follow-the-aircraft subsequence is the one where the crew flies out on the same aircraft as they flew in with. This type of subsequence is very robust with respect to delays, because if crew is delayed on the ingoing flight that can still operate the outgoing flight. Expectations and data from Air New Zealand show that the majority of the optimal subsequences are of this type. A robust subsequence is one in which the time difference between the arrival of the incoming flight and departure of the subsequent flight exceeds the minimum sit time for the crew plus a certain degree of buffer time. Robust subsequences guard against delay propagation just as follow-the-aircraft subsequences. All crew members are entitled to meal breaks at certain time intervals. A meal break subsequence is one which provides crew with the possibility of taking a meal (either in the air or on the ground). Finally, since we typically construct pairings that are longer than three days in generation, it is inevitable that crew will have to overnight somewhere (base or non-base) during this time interval. An overnight subsequence is one in which the crew's idle time on the ground exceeds the minimum rest time.

The above classes of subsequences collectively give the candidate set of subsequences $\mathcal{C}$. We combine the follow-the-aircraft subsequences with each of the other three classes to give three subsets of the candidate set in order to construct networks that allow us to search specifically for robust, meal break, and overnight subsequences. The set $\mathcal{C}^1$ contains subsequence classes 1 and 2, $\mathcal{C}^2$ contains classes 1 and 3, and $\mathcal{C}^3$ consists of classes 1 and 4. We note for completeness that $\bigcup_{k=1}^{3} \mathcal{C}^3 = \mathcal{C}$. To ensure that the shortest path solve on each of the networks is quick, we restrict the number of subsequences for each flight in each of the subsets. That is $|\mathcal{C}^k(f)| \leq n^k$, where n^k is a small integer less than, say, five. The set $\mathcal{C}^1$ is used as the initial limited subsequence set $\mathcal{L}$.

While the pairing generation mirrors what is done in column generation (albeit on slightly different networks), unlike column generation the pairings are not returned directly to the relaxed master problem. Instead, all

negative reduced cost columns are passed through a subsequence analysis phase. For each subsequence $s \in \mathcal{C}$ the analyser maintains four measures that are accumulated over all iterations and updated after analysis of the set of negative reduced cost columns returned by the pairing generators. These statistics are:

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.

The measures are correlated, so a high rank in one measure could also give a high rank in some of the other measures. In fact, three out of the four measures utilise the dual information of the relaxed master problem. Utilising dual information to identify favourable flights is also the topic of Barnhart et al. (1995). Here the authors limit the search to deadhead flights only and incorporate it into a standard column generation procedure.

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.

4 Integer programming framework

Given the preceding section on subsequence generation, one can obtain a high quality solution to the relaxed master problem as shown by Rasmussen et al. (2011). 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)–(8). In this section we present an integer programming framework, which utilises follow-on branching, to force the x_j variables to assume integer values. In Section 4.1 we formalize the methodology of Section 3.2 via a flow-chart, before introducing the idea of constraint branching in Section 4.2. Finally, in Section 4.3, we provide an overview of the complete integer programming approach.

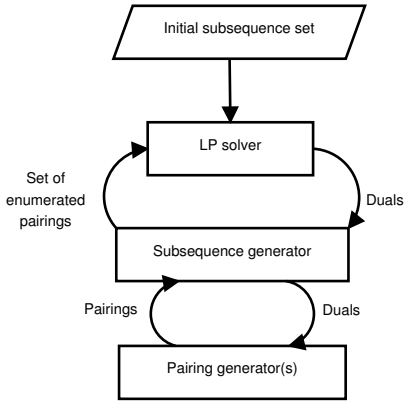

Figure 6: Subsequence generation.

4.1 Solving the LP relaxation

Figure 6 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 $\mathcal{C}^1$ 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. Again, the enumeration is done only with subsequences from the limited set $\mathcal{L}$. The relaxed master problem is then re-optimised with the 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.

4.2 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 consecutively 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. Furthermore, 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 violates 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 respective 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 prevents 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

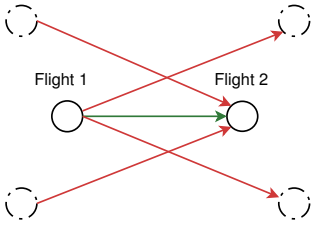

Figure 7: Forcing a subsequence.

ensure that flight 1 and flight 2 are flown consecutively (or not flown at all) in a pairing.

4.3 Solving the integer programme

To produce a high quality solution to Model (1)–(8), we combine the follow-on branching strategy of the preceding section with the subsequence generation methodology of Section 4.1 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, 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 5 we compare our integer solutions to the

	w08r01a	w08r01b	w08r01c	w08r01d	w08r01e	w08r02a	w08r02b	w08r02c	w08r02d	w08r02e	w08r03a	w08r03b	w08r03c	w08r03d	w08r03e	w08r04a	w08r04b	w08r04c	w08r04d
$\lvert\mathcal{F}\rvert$	450	430	430	370	400	380	400	320	450	350	320	420	420	450	320	400	400	350	320
$\lvert\mathcal{B}_1\rvert + \lvert\mathcal{B}_2\rvert$	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.

optimal solution of the relaxed master problem.

5 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 ($|\mathcal{F}|$) and the total number of base constraints for each instance ($|\mathcal{B}_1| + |\mathcal{B}_2|$).

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 be contained in each of the sets $\mathcal{C}^1$, $\mathcal{C}^2$, and $\mathcal{C}^3$?

Based on the results of 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 3.2. 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 $\mathcal{C}^1$, $\mathcal{C}^2$, and $\mathcal{C}^3$. 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.

	Best Integer		Statistics					
instance	obj	time (s)	root LP	cg LP	gap (%)	UF	nodes	time (s)
w08r01a	6.12041e+08	323.53	6.13042e+08	6.02042e+08	1.66	6	105	323.54
w08r01b	3.07034e+08	246.09	3.07598e+08	3.00034e+08	2.33	3	13	246.09
w08r01c	3.05040e+08	634.48	5.06037e+08	3.00031e+08	1.67	3	195	634.55
w08r01d	2.10030e+08	117.34	2.10785e+08	2.03031e+08	3.45	2	5	117.34
w08r01e	8.21038e+08	280.26	8.22034e+08	6.05033e+08	35.70	8	49	280.27
w08r02a	5.00023e+08	19.67	5.00023e+08	5.00021e+08	0.00	5	3	19.67
w08r02b	3.00023e+08	249.96	3.00023e+08	3.00023e+08	0.00	3	47	249.96
w08r02c	8.00019e+08	202.31	8.00521e+08	4.00019e+08	100.00	8	35	202.31
w08r02d	3.00029e+08	468.04	3.00030e+08	3.00025e+08	0.00	3	87	468.06
w08r02e	5.18034e+08	281.61	5.17605e+08	5.12029e+08	1.17	5	41	281.62
w08r03a	8.01024e+08	241.71	8.01023e+08	4.00020e+08	100.25	8	79	241.73
w08r03b	2.11028e+08	184.79	2.12027e+08	2.00034e+08	5.50	2	15	184.79
w08r03c	2.12030e+08	360.95	2.14029e+08	2.02035e+08	4.95	2	77	360.97
w08r03d	3.12030e+08	435.32	3.11531e+08	3.00038e+08	4.00	3	69	435.34
w08r03e	8.19022e+08	102.97	8.19023e+08	4.11028e+08	99.26	8	13	102.97
w08r04a	4.09027e+08	12.69	4.09027e+08	4.01031e+08	1.99	4	1	12.69
w08r04b	4.17025e+08	345.34	4.17026e+08	4.07032e+08	2.46	4	29	345.34
w08r04c	3.16025e+08	8.22	3.16025e+08	3.05033e+08	3.60	3	1	8.22
w08r04d	8.09021e+08	16.29	8.10021e+08	4.00023e+08	102.24	8	3	16.29

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

Table 2 gives the results for the case in which each of the subsequence classes $\mathcal{C}^1$, $\mathcal{C}^2$, and $\mathcal{C}^3$ contains at most three subsequences. For 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 $\mathcal{C}^1$ 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 4.3, 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 $\mathcal{C}^1$, $\mathcal{C}^2$, and $\mathcal{C}^3$ contains at most four subsequences. 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 optimisation 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 undercover more flights than would appear necessary would suggest that a more sophisticated process of including subsequences in the candidate set might be required.

6 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

	Best Integer		Statistics					
instance	obj	time (s)	root LP	cg LP	gap (%)	UF	nodes	time (s)
w08r01a	6.10039e+08	234.22	6.13038e+08	6.02042e+08	1.33	6	15	234.22
w08r01b	3.04036e+08	75.48	3.05202e+08	3.00034e+08	1.33	3	3	75.48
w08r01c	3.02037e+08	1107.07	3.04541e+08	3.00031e+08	0.67	3	25	1107.09
w08r01d	2.09031e+08	1371.26	2.08913e+08	2.03031e+08	2.96	2	115	1371.43
w08r01e	8.19041e+08	295.70	8.21035e+08	6.05033e+08	35.37	8	15	295.71
w08r02a	5.00022e+08	59.37	5.00022e+08	5.00021e+08	0.00	5	7	59.37
w08r02b	3.00023e+08	307.40	3.00023e+08	3.00023e+08	0.00	3	23	307.40
w08r02c	8.00018e+08	48.34	8.00018e+08	4.00019e+08	100.00	8	5	48.35
w08r02d	3.00026e+08	1207.20	3.00026e+08	3.00025e+08	0.00	3	33	1207.22
w08r02e	5.17030e+08	699.73	5.18198e+08	5.12029e+08	0.98	5	23	699.74
w08r03a	8.00021e+08	14.55	8.00021e+08	4.00020e+08	100.00	8	1	14.55
w08r03b	2.06033e+08	1343.80	2.09529e+08	2.00034e+08	3.00	2	171	1344.01
w08r03c	2.08033e+08	1080.62	2.10031e+08	2.02035e+08	2.97	2	71	1080.66
w08r03d	3.11031e+08	19.50	3.11031e+08	3.00038e+08	3.66	3	1	19.50
w08r03e	8.15024e+08	574.02	8.16523e+08	4.11028e+08	98.29	8	41	574.05
w08r04a	4.07028e+08	1333.81	4.07196e+08	4.01031e+08	1.50	4	147	1349.18
w08r04b	4.14028e+08	183.32	4.14029e+08	4.07032e+08	1.72	4	11	183.32
w08r04c	3.13027e+08	12.62	3.13027e+08	3.05033e+08	2.62	3	1	12.62
w08r04d	7.05023e+08	159.30	8.08021e+08	4.00023e+08	76.25	7	15	159.31

Table 3: Classes $\mathcal{C}^1$, $\mathcal{C}^2$, and $\mathcal{C}^3$ have at most four 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 in 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.

Acknowledgements: The authors would like to thank Paul Keating from Air New Zealand for providing the data instances used in this paper.

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, E. Johnson, G. Nemhauser, M. Savelsbergh, and P. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

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, MA, 2nd 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.

J. Clausen, A. Larsen, J. Larsen, and N. J. Rezanova. Disruption management in the airline industry—concepts, models and methods. *Computers & Operations Research*, 37(5):809–821, 2010. Disruption Management.

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.

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. 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.

M. S. Rasmussen, R. M. Lusby, D. M. Ryan, and J. Larsen. Subsequence generation for the airline crew pairing problem. Technical report, Department of Management Engineering, Technical University of Denmark, 2011.

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.

D. M. Ryan and B. Foster. An integer programming approach to scheduling. *Computer Scheduling of Public Transport. Urban Passenger Vehicle and Crew Scheduling. Proceedings of an International Workshop*, pages 269–280, 1981.

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.

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.

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