

Integrated airline crew scheduling: A bi-dynamic constraint aggregation method using neighborhoods

Mohammed Saddoune, Guy Desaulniers, Issmail Elhallaoui and François Soumis

École Polytechnique de Montréal and GERAD

Department of Mathematics and Industrial Engineering

C.P. 6079, Succ. Centre-Ville Montréal, Québec, Canada H3C 3A7

{Mohammed.Saddoune, Guy.Desaulniers, Issmail.Elhallaoui, Francois.Soumis}@gerad.ca

May 9, 2010

Abstract

The integrated crew scheduling (ICS) problem consists of determining, for a set of available crew members, least-cost schedules that cover all flights and respect various safety and collective agreement rules. A schedule is a sequence of pairings interspersed by rest periods that may contain days off. A pairing is a sequence of flights, connections, and rests starting and ending at the same crew base. Given its high complexity, the ICS problem has been traditionally tackled using a sequential two-stage approach, where a crew pairing problem is solved in the first stage and a crew assignment problem in the second stage. Recently, Saddoune *et al.* (2010) developed a model and a column generation/dynamic constraint aggregation method for solving the ICS problem in one stage. Their computational results showed that the integrated approach can yield significant savings in total cost and number of schedules, but requires much higher computational times than the sequential approach. In this paper, we enhance this method to obtain lower computational times. In fact, we develop a bi-dynamic constraint aggregation method that exploits a neighborhood structure when generating columns (schedules) in the column generation method. On a set of seven instances derived from real-world flight schedules, this method allows to reduce the computational times by an average factor of 2.3, while improving the quality of the computed solutions.

Keywords: OR in airlines; crew scheduling; integrated crew pairing and crew assignment; column generation; bi-dynamic constraint aggregation.

1 Introduction

The *airline crew scheduling problem* is one of the most important planning problems faced by the airlines because the total crew cost (salaries, benefits and expenses) is considered, next to the fuel cost, the largest single expense of an airline. Given a schedule of flights (or flight legs) to be operated by the same aircraft fleet, it consists of determining, for a set of available crew members, least-cost schedules that cover all flights and respect various safety and collective agreement rules. For large fleets, this problem is addressed using a two-stage sequential solution approach (see Desaulniers *et al.*, 1998, Gopalakrishnan and Johnson, 2005, and Klabjan, 2005). In the first stage, least-cost crew pairings are built to cover each flight by a single active crew. A pairing is a sequence of one or more duties separated by rest periods and a duty is a sequence of flights separated by connections forming a work day. A pairing must start and end at the same crew base and is assigned to a single crew. In a duty and, thus, in a pairing, a crew can travel as passengers on a flight for repositioning purposes. In this case, the crew is said to be *deadheading* and the flight is called a *deadhead* for this crew. Otherwise, the crew is said to be active. This first stage problem is called the *crew pairing problem*. Given the computed pairings, the second stage problem, called the *crew assignment*

problem, consists of constructing monthly schedules for the available crew members, one crew base at a time. A schedule is a sequence of pairings interspersed by rest periods that may contain days off. Three construction modes are, typically, used for this problem: *bidline* in which anonymous schedules (called bidlines) are built first before being assigned to the crew members according to their preferences and seniority; *rostering* in which personalized schedules are built with the goal of maximizing the preferences of all the employees simultaneously or of balancing as much as possible the workload between the employees; and *preferential bidding* in which personalized schedules are also built but with the goal of maximizing the preferences of the employees in order of seniority. In this paper, we focus on the bidline crew scheduling problem for the pilots. Note that, because the problem definition is the same for the copilots (same objective and constraints), the computed schedules for the pilots can also be assigned to the copilots.

The crew pairing problem has been studied extensively. It is usually formulated as a set partitioning/covering type model and solved using a column generation method embedded into a branch-and-bound procedure (resulting in a branch-and-price method). Column generation is an iterative method that solves alternately a master problem restricted to a subset of the variables and subproblems that generate dynamically variables to add to the restricted master problem (see Desrosiers *et al.*, 1995, Barnhart *et al.*, 1998, Desrosiers and Lübbecke, 2005). Such state-of-the-art methodology was used by Desaulniers *et al.* (1997), Vance *et al.* (1997), Barnhart and Shenoi (1998), Klabjan *et al.* (2001), and Subramanian and Sherali (2008) among others. Literature on the bidline crew assignment problem is rather scant and different versions of the problem were addressed. Jarrah and Diamond (1997), Weir and Johnson (2004), and Boubaker *et al.* (2010) developed mathematical-programming-based algorithms, while Campbell *et al.* (1997) and Christou *et al.* (1999) proposed a simulated annealing and a genetic algorithm, respectively. In particular, Boubaker *et al.* (2010) developed a dynamic constraint aggregation (DCA) method combined with column generation. Introduced by Elhallaoui *et al.* (2005, 2008a), DCA allows to reduce the number of constraints in the column generation master problem (by aggregating clusters of set partitioning constraints) and to compute less fractional linear relaxation solutions, speeding up the overall solution process. With their DCA heuristic, Boubaker *et al.* (2010) produced, in less than one hour of computational time, good quality solutions for instances involving up to 2924 pairings and 564 pilots.

The integration of the crew pairing and the crew assignment problems, called the *integrated crew scheduling (ICS) problem*, was treated in two recent papers. Zeghal and Minoux (2006) proposed

two duty-based integer linear programming models which rely on the assumptions that all duties can be generated a priori and that deadheads can be introduced as needed without any additional costs. Using an exact and a heuristic version of a branch-and-bound algorithm, the authors succeeded to solve only small-sized instances (up to 210 flights or 40 crew members). Saddoune *et al.* (2010) developed a heuristic DCA/column generation method for solving the ICS problem for the pilots in a bidline context. Tested on instances involving up to 7527 flights and 300 pilots, they showed that the solutions derived by their integrated approach yield substantial savings (3.37% on the total cost and 5.54% on the number of pilots) when compared to the solutions computed by a traditional sequential approach. The main challenge with the ICS problem is the computational time required for solving it: Saddoune *et al.* (2010) report solving a small instance (1101 flights and 30 pilots) in 747 minutes using column generation alone. With their DCA method, these authors succeeded to solve the same instance in only 27 minutes. Despite this substantial speedup, the computational times remain quite high compared to those obtained with the sequential approach as they were, on average, 6.8 times higher in Saddoune *et al.* (2010).

The main goal of this paper is to enhance the integrated solution approach of Saddoune *et al.* (2010) for the ICS problem so as to reduce its computational times. To do so, we propose to use another variant of the DCA method called the bi-dynamic DCA (BDCA) method and developed by Elhallaoui *et al.* (2008b). Besides reducing the size of the column generation master problem as in the DCA method, the BDCA method also reduces the size of the networks in the column generation subproblems. This reduction is performed by removing arcs from the networks, resulting in a partial pricing strategy. In Elhallaoui *et al.* (2008b), the selection of the arcs to remove is based on dual values obtained from the master problem. Here, we propose to use a selection criterion involving reduced costs. Furthermore, the arc selection procedure is limited by a neighborhood that can vary from one column generation to another. Such a neighborhood strategy allows to exploit the problem structure and to focus the search for new columns that, when combined together, have a high probability of provoking a decrease of the objective function value. The resulting method is called the BDCA method with neighborhoods and abbreviated by BDCA-N. Through computational experiments on seven instances derived from real-life flight schedules, we show that this BDCA-N method yields computational times that are, on average, 2.3 times smaller than those of the DCA method of Saddoune *et al.* (2010). Furthermore, our results indicate that larger savings on the total cost (on average, 4.02% when compared to the total cost obtained by the sequential approach) can be achieved with the BDCA-N method.

The rest of this paper is organized as follows. Section 2 provides a detailed definition of the ICS problem considered. Section 3 formulates this problem as a set partitioning type model, whereas Section 4 describes a generic version of the BDCA-N method and its specialization for the ICS problem. Computational results are reported in Section 5, before drawing some conclusions in Section 6.

2 Problem statement

The definition of the ICS problem (for the pilots in a bidline context) can vary from one airline to another because it depends on the collective agreement of the pilots. Here, we consider the same definition as in Saddoune *et al.* (2010) that combines the crew pairing problem definition of Mercier *et al.* (2005) and Saddoune *et al.* (2009) and the bidline crew assignment problem definition of Boubaker *et al.* (2010). It includes the most important features that are common to most airlines. Given a set of flights operated by the same aircraft type over a planning horizon (typically, one month), a set of crew bases, and the number of pilots available per crew base, the ICS problem consists of finding least-cost anonymous feasible schedules such that each flight is assigned to exactly one active pilot and the pilot availability per base holds as much as possible. Pilot availability does not include the pilots expected to be in reserve. If insufficient, it can be exceeded at a high penalty cost, meaning that less pilots will be in reserve.

A schedule is feasible if it satisfies the following safety and collective agreement rules. Between every pair of consecutive pairings, there must be a rest, called a post-courrier rest, whose duration is greater than or equal to a minimum duration. This rest can include one or several days off (from midnight to midnight). A schedule is subject to a minimum number of days off, a maximum number of consecutive working days, and a maximum credited flying time, where credited flying time typically corresponds to the active flying time plus 50% of the deadhead flying time. A pairing cannot exceed a maximum duration and cannot contain more than a maximum number of duties. Furthermore, each duty in each pairing must contain a briefing and a debriefing period, and is subject to a maximum number of landings, a maximum working time, and a maximum total duration. Finally, there must be a minimum connection time between each pair of consecutive flights in a duty.

A schedule incurs a fixed and a variable cost. The fixed cost includes all fees (except the salary) paid by the airline for the pilot such as trainings, vacations and overhead, while the variable cost depends on the pairings included in the schedule. The cost of a pairing is a very complex function

of the pairing duration, the durations of the duties it contains, and the deadheads flown in the pairing among others. Given its high complexity, this function is approximated as in Saddoune *et al.* (2009, 2010). Because each flight must be covered by an active pilot, a large portion of the total pairing cost is fixed and can be omitted from the problem. The rest of the cost of a pairing concerns the deadhead costs, a guaranteed minimum credited flying time paid per duty, and the durations of the pairing and its duties. These durations highly depend on the connection and rest times arising in the pairing. These times are, hereafter, referred to as waiting periods.

The cost of a schedule s is defined by

$$c_s = k + \sum_{p \in P_s} c_p \quad (1)$$

where k is the schedule fixed cost, P_s is the set of pairings composing this schedule, and c_p is the cost of pairing p . The cost of a pairing p that contains a set of duties D_p , a set of deadheads H_p , and a set of waiting periods W_p is given by:

$$c_p = \sum_{w \in W_p} g(\delta_w) + \sum_{h \in H_p} (\gamma + \mu \delta_h) + \nu \sum_{d \in D_p} \max\{0, V_{min} - v_d\}, \quad (2)$$

where δ_w is the duration of waiting period w , $g(\cdot)$ is a waiting cost function, γ is a fixed cost for each deadhead, δ_h is the duration of deadhead h , μ is a unit cost for each minute spent deadheading, V_{min} is the guaranteed minimum credited flying time paid per duty, v_d the total credited flying time in duty d , and ν the salary paid for each flying hour. The waiting cost function $g(\cdot)$ is the one proposed by Mercier *et al.* (2005) and used by Saddoune *et al.* (2009, 2010). It is a piecewise linear function that penalizes too short connections for increased robustness, and too long connections and rests for reduced pairing cost. See one of the above references for more details.

3 Mathematical model

To formulate the problem, we use the following notation.

F : set of flights to cover;

B : set of crew bases;

q_b : number of pilots available at base b (excluding those expected to be in reserve);

β : penalty cost for each additional pilot (that should reflect the expected cost of having one less pilot in reserve);

S^b : set of feasible schedules for pilots at base b ;

c_s : cost of schedule s (as defined by (1));

a_{fs} : binary parameter equal to 1 if flight f is actively covered in schedule s and 0 otherwise;
 x_s : binary variable taking value 1 if schedule s is selected and 0 otherwise;
 y_b : surplus variable indicating the number of extra pilots required at base b .

As proposed by Saddoune *et al.* (2010), the ICS problem can be formulated as the following set partitioning type model:

$$\text{Minimize } \sum_{b \in B} \sum_{s \in S^b} c_s x_s + \beta \sum_{b \in B} y_b \quad (3)$$

$$\text{subject to: } \sum_{b \in B} \sum_{s \in S^b} a_{fs} x_s = 1, \quad \forall f \in F \quad (4)$$

$$\sum_{s \in S^b} x_s - y_b \leq q_b, \quad \forall b \in B \quad (5)$$

$$x_s \in \{0, 1\}, \quad \forall b \in B, s \in S^b \quad (6)$$

$$y_b \geq 0, \quad \forall b \in B. \quad (7)$$

The objective function (3) minimizes the sum of the schedule costs and the penalty costs for scheduling additional pilots. Flight coverage is imposed by the set partitioning constraints (4), whereas soft pilot availability at each base is ensured by constraints (5). Binary requirements on the x_s variables are given by (6). These requirements imply integrality on the y_b variables that can, thus, be only subject to nonnegativity constraints (7).

4 Solution methods

Saddoune *et al.* (2010) developed a DCA method for solving model (3)–(7). We propose to solve it using four different variants of the BDCA method introduced by Elhallaoui *et al.* (2008b). All these variants are combined with column generation for solving linear relaxations and embedded into a variable fixing procedure for deriving integer solutions.

4.1 Column generation

In practice, model (3)–(7) contains a huge number of variables x_s , one per feasible schedule. To avoid enumerating all these variables, a column generation method can be applied for solving the linear relaxation of this model, which is called the master problem in this case. Such a method (see Desrosiers *et al.*, 1995, Barnhart *et al.*, 1998, Desrosiers and Lübbecke, 2005) is iterative and solves at each iteration a restricted master problem (RMP) and one or several subproblems. The RMP is the master problem restricted to a subset of its x_s variables and all y_b variables. Solving it provides

a primal and a dual solution. Given this dual solution, the role of the subproblems is to determine whether or not there exist negative reduced cost x_s variables (columns) among those that are not considered in the current RMP. If none exist, the current RMP primal solution is optimal for the master problem and the algorithm stops. Otherwise, negative reduced cost columns identified by the subproblems are added to the RMP before starting a new iteration.

For the ICS problem, there is one subproblem per crew base that aims at finding a feasible schedule with least reduced cost for the associated base. This subproblem is a shortest path problem with resource constraints defined on an acyclic time-space network that represents implicitly all feasible schedules for this base. The resource constraints are needed to restrict schedule feasibility and compute schedule reduced cost. Such a subproblem can be solved by a label-setting algorithm (see Desrosiers *et al.*, 1995, Irnich and Desaulniers, 2005).

Because we use the same networks and resources as in Saddoune *et al.* (2010), we only summarize them here¹ and refer the reader to Saddoune *et al.* (2010) for further details. Such a time-space network contains six node types, including a *source* and a *sink* node. It involves eleven arc types: *start of schedule*, *end of schedule*, *start of duty* and *start of pairing* arcs; *flight* and *deadhead* arcs to represent the assignment of a pilot to an active or a deadhead flight, respectively; *rest* arcs to model rests between two duties; *waiting* arcs to extend such a rest or to model the connection between two flights in a same duty; *post-pairing* and *post-courrier* arcs to represent rests between two pairings with or without a day off; and, finally, *day off* to allow the extension of these rests.

Every feasible schedule for the associated base corresponds to a path from the source to the sink node in this network. On the other hand, not all paths represent a feasible schedule. Most schedule feasibility rules are treated during path construction in the label-setting algorithm via the use of constrained resource variables. A resource is a quantity that varies along a path and whose value is restricted to fall within a given interval, called a resource window, at each node. For the ICS problem, we use a total of nine resources. A first set restricts the feasibility of the pairings and comprises one resource to model each of the following constraints: maximum pairing duration, maximum number of duties in a pairing, maximum number of landings per duty, maximum working time per duty, and maximum total duty duration. A second set of three resources deals with the following schedule feasibility constraints: minimum number of days off, maximum number of

¹Comment for the reviewers: In Saddoune *et al.* (2010), the description of the networks and resources required three complete pages, including a figure spanning two-thirds of a page. Thus, for reasons of conciseness, we have opted here for a short summary that obviously does not contain all the details. We believe that, to give more details, we should provide a figure and describe it, which would take too much space.

consecutive working days, and maximum credited flying time. Finally, a ninth resource is required to compute the penalty incurred when the guaranteed minimum credited flying time per duty is not reached.

4.2 Dynamic constraint aggregation

4.2.1 Basic concepts

In practice, because the average number of flights per schedule is relatively large, the column generation method suffers from high degeneracy and can be inefficient for solving the linear relaxation of (3)–(7). To overcome this difficulty, Elhallaoui *et al.* (2005) proposed DCA that can be combined with column generation. In this case, a DCA method uses an aggregated restricted master problem (ARMP) that is obtained by aggregating clusters of the RMP set partitioning constraints and keeping one representative constraint for each cluster. This constraint aggregation can change from one column generation iteration to another to guarantee the exactness of the method. Since each set partitioning constraint (4) is associated with a flight, a cluster corresponds to a non-empty subset of flights and an aggregation is performed according to a partition Q of the flights in F into clusters. The solution process starts using an initial partition that can be computed from a heuristic solution of the problem or from logical reasoning. A variable x_s is compatible with partition Q if the set of flights covered by the corresponding schedule is the union of some clusters in Q . Otherwise, this variable is incompatible. The ARMP only contains compatible x_s variables (and all y_b variables). Once solved, it provides a primal and an aggregated dual solution. To allow pricing all (compatible and incompatible) x_s variables, this dual solution is disaggregated using a repetitive shortest path procedure to yield disaggregated dual values, that is, one dual value for each set partitioning constraint of the original model. A newly generated variable compatible with the current partition can be added to the ARMP without modifying the current partition Q . At the opposite, an incompatible variable cannot be added to the ARMP without modifying it. By working with a reduced sized master problem, DCA reduces the impact of degeneracy and speeds up the computational time per column generation iteration.

To maintain a higher level of aggregation during the solution process, Elhallaoui *et al.* (2008a) developed a partial pricing strategy that favors the generation of columns that are compatible or slightly incompatible with the current partition Q . To do so, they define a number of incompatibilities that a variable x_s can have with respect to the current partition. This number estimates the number of additional clusters needed in the partition to make the variable compatible. This

strategy gives rise to the multi-phase DCA (MPDCA) method which executes a sequence of phases where, in a phase number k , only the variables with a number of incompatibilities less than or equal to k are priced out (this constraint is handled by an additional resource in the subproblems). The algorithm is exact if the last phase number k is sufficiently large to ensure the pricing of all feasible columns. The MPDCA method favors a slow disaggregating process of the ARMP (when needed) and, thus, speeds up the overall solution process by keeping the size of the ARMP relatively small. The DCA method used by Saddoune *et al.* (2010) was, in fact, an MPDCA method.

In Elhallaoui *et al.* (2008b), the same authors proposed an enhanced MPDCA method, called the bi-dynamic constraint aggregation (BDCA) method, that reduces the size of the subproblem networks besides reducing the size of the RMP. This network reduction can vary from one column generation iteration to another and proceeds as follows. First, clusters are selected in increasing order of their corresponding set partitioning dual values until reaching a predetermined number of clusters. Then, every arc that would force the breaking of a selected cluster is removed from all networks (see Section 4.2.4). The number of clusters to select is set to $\lfloor |Q| \cdot RL \rfloor$, where the reduction level RL is a parameter whose value belongs to $[0, 1]$. An RL value close to 1 yields highly reduced networks. At the opposite, the networks are almost complete when RL takes a value near 0. With this reduction strategy, the flights of a selected cluster are either all included in a generated column or none of them is included, that is, this cluster remains aggregated. Solving the subproblems with sufficiently reduced networks is much faster than with complete networks and further favors the generation of compatible and slightly incompatible columns. However, when no negative reduced cost columns can be generated from the reduced subproblems, the subproblems with complete networks are solved to ensure the exactness of the overall method. As the multi-phase strategy, this network reduction strategy corresponds to a partial pricing strategy.

In the BDCA method of Elhallaoui *et al.* (2010), the cluster selection procedure does not take advantage of the problem structure. Selected clusters are chosen solely on the basis of dual values and may not offer much possibilities to generate columns that can be combined for improving the current RMP primal solution. In this paper, we propose to exploit the problem structure by using a neighborhood for restricting the cluster selection procedure. A function is used to determine whether or not a given cluster belongs to this neighborhood. Clusters that can be broken are chosen in priority in this neighborhood. During the solution process, several neighborhoods are used but each neighborhood is kept for a certain number of consecutive column generation iterations to increase the possibility of generating columns with high interaction in these iterations.

Algorithm 1 : BDCA-N

```
1: Select an initial neighborhood  $N$ 
2: Set  $k := 0$ ,  $Z_{old} := \infty$ ,  $Z := \infty$ 
3: Create an initial partition  $Q$  and build the ARMP
4: Solve the ARMP with artificial variables
5: repeat
6:   Select a subset of clusters  $U$  according to  $RL$  and  $N$ 
7:    $negRedCostCols := LocalSearch(Z, Q, k, U)$ 
8:   if  $negRedCostCols = false$  then
9:      $k := k + 1$ 
10:  else if  $Z_{old} - Z \leq \delta$  then
11:    Select a new neighborhood  $N$ 
12:     $Z_{old} := Z$ 
13: until  $k > k_{max}$ 
```

4.2.2 Generic BDCA-N algorithm

In this section, we present a generic version of the BDCA-N algorithm. Its specialization to the ICS problem will be discussed in a subsequent section. The pseudo-code of this algorithm is given in Algorithm 1 and commented upon in the next paragraphs.

In Steps 1 to 4, the algorithm is initialized. In particular, it starts in phase $k = 0$ and sets to infinity the value of the current RMP solution (Z), and the value of the RMP solution computed in the last iteration using the previous neighborhood (Z_{old}). In Step 4, artificial variables with very large costs are added to the ARMP to ensure that a primal and an aggregated dual solution can be obtained even if not enough columns have been generated yet.

In Step 6, clusters that are forced to remain aggregated are selected according to the reduction level RL and the current neighborhood N . The selection procedure creates two lists of clusters: the first one contains all clusters in N and the second one all the others. Both lists are then sorted in increasing order of their aggregated dual values (a different criterion can be used). Finally, the clusters are selected in order in the first list and, if needed, in the second list until reaching the predetermined number of clusters to select (as explained above, this number is set to $\lfloor |Q| \cdot RL \rfloor$). The set of selected clusters is denoted by U .

In Step 7, a local search procedure, which is detailed below, is called. Essentially, this procedure performs column generation iterations in the DCA framework under a restriction (imposed through the current set U) on the columns that can be generated. Because the number of column generation

Algorithm 2 : $LocalSearch(Z, Q, k, U)$

```
1:  $curIter := 1$ 
2: repeat
3:   Compute disaggregated dual values
4:    $redNetwork := true$ 
5:   Reduce the subproblems according to  $U$  and solve them in phase  $k$ 
6:   if negative reduced cost columns found then
7:     Update partition  $Q$  (if needed) and the ARMP
8:     Solve the ARMP and update  $Z$ 
9:      $curIter := curIter + 1$ 
10:  else if  $redNetwork = true$  and  $k > 0$  then
11:     $redNetwork := false$ 
12:    Solve the complete subproblems in phase  $k$  and go to Step 6
13:  else
14:    return false
15: until  $curIter > maxIter_k$ 
16: return true
```

iterations in this procedure is limited, the procedure returns a boolean value equal to *true* if negative reduced cost columns were generated in the last iteration and *false* otherwise. This value is assigned to the variable $negRedCostCols$.

If no columns were generated in this last iteration (Step 8), then the phase number k is increased by one in Step 9. Otherwise, the neighborhood is changed (Step 11) if the current neighborhood did not yield a sufficient decrease of the RMP objective value (Z) in procedure $LocalSearch$. In this test (Step 10), parameter δ takes a predefined positive small value. When negative reduced cost columns were generated in the last column generation iteration and a sufficient decrease in the RMP objective value was observed, the current neighborhood is kept. In this case, new clusters are selected when returning to Step 6 and the search using this neighborhood is intensified by applying again the $LocalSearch$ procedure.

Steps 6 to 12 are repeated until the phase number exceeds k_{max} , the maximum phase number. This maximum phase number can always be set to a value that ensures the exactness of the method.

The pseudo-code of procedure $LocalSearch$ is given in Algorithm 2. It executes column generation iterations using the same neighborhood: subproblems are solved in Steps 5 and 12, and the ARMP is solved in Step 8. In Step 3 of this procedure, disaggregated dual values are computed to be able to also price incompatible columns (see Elhallaoui *et al.*, 2005, for details). In Step 5, the

subproblem networks are first reduced according to the subset of clusters U and then solved. If negative reduced cost columns are found, they are used to update partition Q if needed before being added to the ARMP, which is then solved in Step 8. In an update, partition Q can be disaggregated and aggregated. It is disaggregated when the ratio of the least reduced cost of the generated incompatible variables over that of the generated compatible variables exceeds a given threshold. It is aggregated after such a disaggregation if the resulting partition contains too many clusters. When no negative reduced cost columns are generated using the reduced networks and the phase number is positive, the subproblems are solved again in Step 12 using the complete networks before returning to Step 6. This guarantees the exactness of the method. Note that, in phase $k = 0$ where it is not possible to generate incompatible columns, there is no need to solve the subproblems with complete networks as they are equivalent to the reduced subproblems.

When both reduced and complete subproblems fail to generate negative reduced cost columns, the procedure stops (Step 14). The procedure can also stop prematurely in Step 15 when a predefined maximum number of iterations ($maxIter_k$) that depends on the phase number k is reached. This premature halt allows to diversify the search by changing the current neighborhood N or at least the selected cluster subset U .

4.2.3 BDCA, MPDCA, and DCA algorithms

The BDCA, MPDCA and DCA algorithms of Elhallaoui *et al.* (2005, 2008a, 2008b) are special cases of the BDCA-N algorithm. The BDCA algorithm of Elhallaoui *et al.* (2008b) is obtained by setting $maxIter_k = 1$ for all phases $k > 0$ and considering throughout the solution process a single neighborhood N that contains all possible clusters. In this case, a single column generation iteration is performed in procedure *LocalSearch* and a new subset U of clusters is selected for each iteration. Furthermore, Steps 10 to 12 of Algorithm 1 can be omitted together with the use of Z and Z_{old} . For our computational experiments, we used two variants of this BDCA algorithm. The difference between two variants concerns how the clusters are selected in Step 6 of Algorithm 1 as explained below.

The MPDCA algorithm of Elhallaoui *et al.* (2008a) is a special case of the BDCA algorithm in which the reduction level RL is set at 0, that is, the subset U always contains no clusters and the subproblem networks are always complete. In this case, Steps 4 and 10 to 12 can be removed from Algorithm 2 to avoid solving the same subproblems twice per iteration. This MPDCA algorithm was used by Saddoune *et al.* (2010).

Finally, the original DCA algorithm of Elhallaoui *et al.* (2005) is a special case of the MPDCA algorithm that involves a single phase with number $k = k_{max}$, that is, the phase number k should be initially set to k_{max} in Step 2 of Algorithm 1.

4.2.4 Specialization of the BDCA-N algorithm for the ICS problem

To specialize the BDCA-N algorithm for the ICS problem, we need to define the initial partition Q , the number of incompatibilities of a variable x_s , and the neighborhoods N .

An initial partition Q is composed of disjoint clusters of flights. As shown in Elhallaoui *et al.* (2008a), the quality of this partition can greatly influence the performance of the algorithm. It is considered good if the clusters contain flights that have a high probability of being actively covered by the same schedule in the solution computed for the linear relaxation of (3)–(7). As in Saddoune *et al.* (2010), we construct this initial partition by solving the crew pairing problem using the rolling horizon/column generation approach developed by these authors. The flights actively covered in each pairing of the computed solution form a cluster in partition Q . Note that, in this case, building the initial partition corresponds to executing the first stage of the sequential solution approach.

Pricing in phase k is restricted to variables that have at most k incompatibilities with the current partition Q . The definition of the number of incompatibilities of a variable is problem specific. As mentioned earlier, this number should estimate the minimum number of additional clusters needed to make the variable compatible. Here, we use the same definition as in Saddoune *et al.* (2010) which requires to slightly modify the subproblem networks and to associate with each arc a number of incompatibilities (0, 1 or 2). The number of incompatibilities of a variable x_s is given by the sum of the numbers of incompatibilities of the arcs in the path representing schedule s . Figure 1 illustrates the number of incompatibilities of different arcs (this number is written beside the arc if equal to 1 or 2). It shows the flight arcs composing three clusters in Q and examples of other arc types (e.g., waiting, rest, deadhead, day off, etc.) that were part of the networks used for column generation without DCA (see Section 4.1). In addition to these arcs, the networks contain *intra-cluster* arcs that directly link the flight arcs of each pair of consecutive flights in a cluster. Such an arc represents all the activities (wait, rest, or deadhead) performed between these two flights. The number of incompatibilities of an arc that is not an intra-cluster arc is equal to 2 if the following two conditions hold: i) its initial node is the arrival node of a flight that is not the last of its cluster, and ii) its final node is the departure node of a flight that is not the first of its cluster. It is equal to 1 if the arc is not an intra-cluster arc and only one of these two conditions holds. It is equal to 0

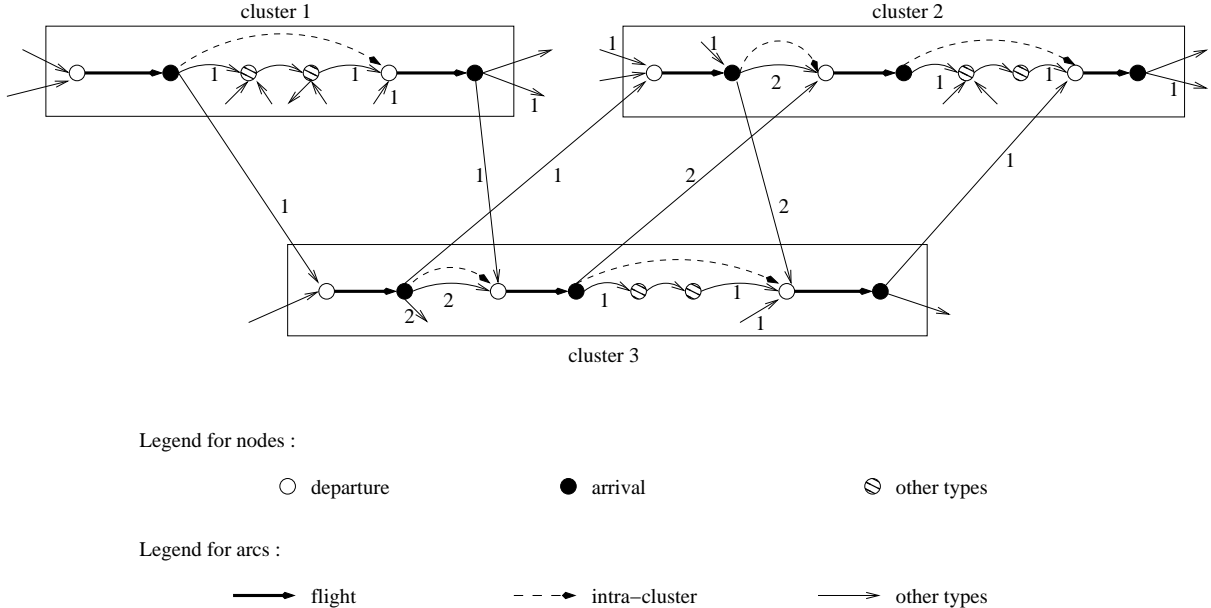


Figure 1: Number of incompatibilities associated with the arcs

for all the other arcs (including all intra-cluster arcs). Computing the incompatibilities in this way clearly favors the generation of paths (columns) that use intra-cluster arcs during the first phases of the BDCA-N algorithm, slowing down the disaggregation process of the ARMP.

Figure 1 also allows to better understand which arcs are eliminated from the networks reduced in Step 5 of Algorithm 2. For instance, if a cluster, say cluster 2, belongs to U , then all arcs with a positive number of incompatibilities that are incident to an inner arrival or departure node of cluster 2 are removed from the networks. In this way, any path using the first flight arc of this cluster must necessarily go through the alternate sequence of flight and intra-cluster arcs in this cluster, preserving its aggregation. For our tests, the reduction level parameter RL was set to 0.93 (that is, arcs were removed for 93% of the clusters), yielding highly reduced networks.

To increase the chances of generating complementary columns (that is, columns that can provoke a decrease of the objective function value) using the reduced subproblem networks, a neighborhood should contain flights that offer flight exchange possibilities between the constructed schedules. These flights should, therefore, operate approximately at the same time. Consequently, we propose to use a fixed set of neighborhoods that are defined as follows. First, the (one-month) planning horizon is divided into time slices of equal length (L days), each one overlapping with the next one by one day. Note that the last slice might be shorter than L days. For example, if $L = 9$ and the horizon has 31 days, then the time slices are $[1, 9]$, $[9, 17]$, $[17, 25]$, and $[25, 31]$. Let T be the set

of time slices and number them in chronological order from 1 to $|T|$. A neighborhood N_t , $t \in T$, is then associated with each time slice and a flight cluster is said to belong to a neighborhood N_t if at least one of its flights operates in time slice $t \in T$. In the BDCA-N algorithm, we chose $N_{|T|}$ as an initial neighborhood (Step 1 of Algorithm 1). When changing neighborhood in Step 11, we select N_{t-1} as the new neighborhood if the current neighborhood is N_t (where $N_0 = N_{|T|}$). In Algorithm 2, each neighborhood N is explored a maximum number of iterations $maxIter_k$ that depends on the current phase number k . In phase $k = 0$, $maxIter_k$ is set to a very large value because the current neighborhood has no influence on the subproblems (only compatible variables can be generated). In all other phases, $maxIter_k$ is set to the same value (10 in our tests).

Given the high complexity of the ICS problem, we apply the following two heuristic stopping criteria when solving a linear relaxation with a variant of the BDCA-N algorithm. First, the value of k_{max} is set to 1, allowing the use of only phases $k = 0$ and $k = 1$. Second, column generation is stopped when the objective function value decreases by less than a given threshold $minD$ in itD consecutive iterations, where, in our tests, $minD$ was set to 0.01% and itD to 20 (resp. 10) for the small-sized (resp. medium-sized) instances. These stopping criteria were also used by Saddoune *et al.* (2010). However, during preliminary computational experiments, we observed that the column generation process often stops prematurely in phase 0 with an objective value that is sometimes relatively far from the optimal value. In consequence, we also devised and tested a variant of the BDCA-N algorithm, denoted BDCA-N-1, that pursues in phase 1 when such a premature halt occurs in phase 0.

4.2.5 Algorithm variants

For our computational experiments, we considered four variants of the BDCA-N algorithm (in fact, two are special cases). The first variant is a direct adaptation to the ICS problem of the BDCA algorithm introduced by Elhallaoui *et al.* (2008b), in which the clusters in U are selected using a criterion based on their aggregated dual values. The second variant corresponds to the first variant except that the clusters in U are rather selected using an approximate reduced cost criterion. The approximate reduced cost \bar{r}_ℓ of a flight cluster ℓ is given by $\bar{r}_\ell = \sum_{w \in W_\ell} g(\delta_w) + \sum_{h \in H_\ell} (\gamma + \mu \delta_h) - \alpha_\ell$, where H_ℓ and W_ℓ are the sets of deadhead flights and waiting periods occurring between the flights of cluster ℓ , respectively, and α_ℓ is the dual value of the aggregated set partitioning constraint (4) associated with cluster ℓ in the ARMP. This reduced cost is approximate because it does not consider the guaranteed minimum credited flying time paid per duty that cannot be determined

for incomplete duties. In practice, this approximation is very good because, for most duties, this omitted cost is equal to zero. To build U , the selection procedure first order the clusters in decreasing order of their reduced cost and then selects the first $\lfloor |Q| \cdot RL \rfloor$ clusters. Finally, the third and fourth variants are the complete BDCA-N and BDCA-N-1 algorithms (with neighborhoods) that rely on the reduced cost criterion for selecting the clusters in U . These four variants are denoted BDCA-DV (for dual values), BDCA-RC (for reduced costs), BDCA-N, and BDCA-N-1, respectively.

4.3 Variable fixing procedure

To derive an integer solution, a variable fixing procedure that imposes decisions permanently is used. This procedure creates a branch-and-bound search tree with a single branch in which column generation is applied at every node to compute a new linear relaxation solution. The tree exploration stops as soon as one of these solutions is integer. Two types of decisions are considered. In priority, we fix to 1 all x_s variables taking a fractional value greater than a predetermined threshold (0.75 for our tests). As a secondary option, we impose that two flights be assigned to the same schedule and performed consecutively. Such inter-task (or follow-on) constraints are treated directly in the subproblems (see Irnich and Desaulniers, 2005) and correspond to fixing at 0 all x_s variables whose schedule s covers only one of the two flights or both flights but not consecutively.

5 Computational experiments

We conducted computational experiments to assess and compare the efficiency of the proposed algorithms. These tests were carried out on seven instances ($I1$ to $I7$) derived from a one-month flight schedule that was operated by a major North American airline. Each instance corresponds to a specific short- or medium-haul aircraft type. All these instances involve three crew bases, between 1011 and 7527 flights, and between 26 and 54 stations as reported in Table 1.

Instances $I1$ to $I7$ are the same as those used in Saddoune *et al.* (2010). Table 1 also provides a summary of their computational results that will serve as benchmark results. Saddoune *et al.* (2010) compared the traditional two-stage sequential (SEQ) approach with the MPDCA approach that they developed and that was described above. For each instance, we report the total computational time of the SEQ approach (in minutes) as well as the total cost and the total number of schedules in the solution that it produced. Furthermore, for the MPDCA approach, we indicate, with respect to the results of the SEQ approach, the increase factor of the computational time ($CPU\ ratio\ MPDCA/SEQ$), the savings on the total cost (in percentage), and the savings on the number

Instance	Flights	Stations	SEQ approach			MPDCA approach		
			CPU	Cost	No. sched	CPU ratio MPDCA/SEQ	Savings (%)	
			(min)				Cost	Sched
<i>I1</i>	1011	26	4.0	767754	33	6.9	4.18	9.09
<i>I2</i>	1463	35	5.8	957989	34	5.6	4.63	8.82
<i>I3</i>	1793	41	11.4	1313391	47	12.7	3.27	8.51
<i>I4</i>	5466	49	522.6	3502527	145	2.9	2.80	4.82
<i>I5</i>	5639	34	231.9	4835090	247	8.6	2.85	2.02
<i>I6</i>	5755	52	260.0	5144122	223	5.9	4.82	4.93
<i>I7</i>	7527	54	507.6	6536094	305	5.2	1.05	0.65
Average						6.8	3.37	5.54

Table 1: Instances and results of Saddoune *et al.* (2010)

of schedules (in percentage). These results clearly show that integrating crew pairing and crew assignment can yield significant savings (on average, 3.37% on the total cost and 5.54% on the number of schedules) at the expense of much higher computational times (on average, 6.8 times higher) with the MPDCA approach. In the following, we compare these results to those obtained by the proposed BDCA-N variants.

All tests were performed on a linux PC machine (equipped with an Intel Xeon processor clocked at 2.8 GHz). Our implementations are coded in C++ and use the GENCOL column generation library (version 4.5) and the CPLEX linear programming solver (version 10.1) for solving the aggregated restricted master problems. Saddoune *et al.* (2010) used the exact same setup.

5.1 Results for the BDCA-DV and BDCA-RC variants

First, we tested the BDCA-DV and BDCA-RC variants. Table 2 summarizes the computed results. As for the MPDCA approach, we report CPU time increase factors, total cost savings and number of schedule savings with respect to the SEQ approach.

From these results, we make the following observations. First, using BDCA helps to reduce significantly the computational times. Indeed, when compared to the MPDCA approach of Saddoune *et al.* (2010), the BDCA-DV (resp. BDCA-RC) approach yields an average speedup factor of 2.6 (resp. 2.3), that is, the CPU ratio drops from 6.8 to 2.6 (resp. 2.9). These approaches remain, however, slower than the SEQ approach. We also observe that the solution quality has deteriorated with these speedups (from average cost savings of 3.37% for the MPDCA algorithm to average cost savings of 2.63% for the BDCA-DV algorithm or 2.96% for the BDCA-RC algorithm). This dete-

Instance	BDCA-DV approach			BDCA-RC approach		
	CPU ratio	Savings (%)		CPU ratio	Savings (%)	
	BDCA-DV/SEQ	Cost	Sched	BDCA-RC/SEQ	Cost	Sched
<i>I1</i>	1.8	2.14	3.03	1.5	3.76	6.06
<i>I2</i>	2.4	3.40	8.82	2.3	3.25	5.88
<i>I3</i>	2.2	2.68	8.51	2.2	2.54	8.51
<i>I4</i>	1.8	2.89	4.79	1.6	3.26	4.79
<i>I5</i>	4.3	1.61	1.61	6.5	2.05	2.02
<i>I6</i>	2.9	5.75	5.38	2.9	5.77	5.82
<i>I7</i>	2.9	-0.04	0.32	3.4	0.09	0.65
Average	2.6	2.63	4.63	2.9	2.96	4.81

Table 2: Results for the BDCA-DV and BDCA-RC variants

rioration is due to very premature halts of the column generation process. Indeed, when working with reduced networks, less columns are generated on average at each iteration and the objective function value decreases more slowly from one iteration to another, increasing the chances of a premature halt.

Comparing together the results of both BDCA-DV and BDCA-RC variants, we remark that, on average, the former approach is slightly faster than the latter approach, but produces solutions that are slightly costlier. The outcome varies, however, from one instance to another. It is, therefore, difficult to determine which of these two variants is the best. Given that the BDCA-RC variant yielded, on average, an additional 0.33% in cost savings (from 2.63% to 2.96%) for an average difference of 0.3 in the CPU time ratio (from 2.6 to 2.9), we decided to use the reduced cost criterion in the BDCA-N and BDCA-N-1 variants.

5.2 Results for the BDCA-N and BDCA-N-1 variants

As mentioned in Section 4.2.4, neighborhoods are used to increase the chances of generating complementary columns. For the ICS problem, they concentrate the flight clusters that can be disaggregated in the same time slice. Therefore, given a fixed number of clusters that can be disaggregated, the proportion of them in a time slice depends on the time slice length. A very long time slice is equivalent to not using neighborhoods, whereas a very short time slice highly restricts the cluster choice to those intersecting with the time slice, neglecting the reduced cost criterion. To determine the time slice length L , we solved instances *I1* to *I7* using the BDCA-N algorithm with different values of L , namely, $L = 7, 9, 11, 16, 31$. These values were chosen because they are the smallest values that yield, respectively, 5, 4, 3, 2, and 1 time slices (with a one-day overlap between two

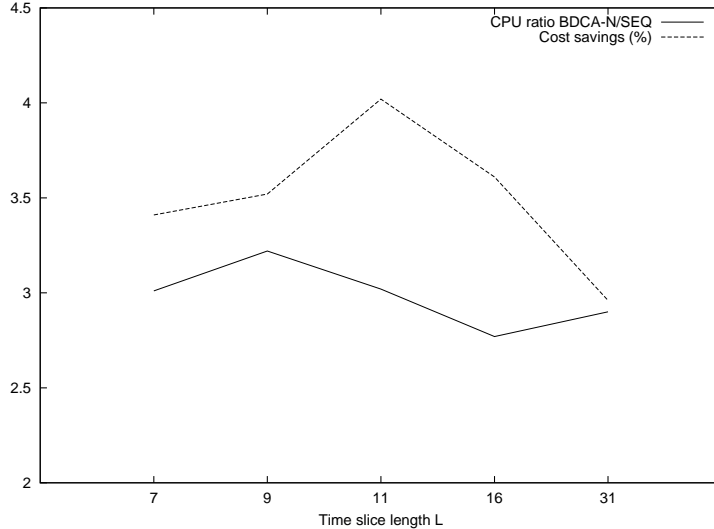


Figure 2: Average results of the BDCA-N algorithm for various time slice lengths

consecutive slices) covering a 31-day month. The results of these experiments are presented in Figure 2, where the horizontal axis indicates the value of L . The full line gives the linear interpolation of the computed points $(L, R(L))$, where $R(L)$ is the average CPU ratio BDCA-N/SEQ when L is used as the time slice length. Similarly, the dotted line provides the linear interpolation of the points $(L, C(L))$, where $C(L)$ is the average total cost savings. These results show that using too short or too long time slices is not as good as using time slices of medium length. In fact, the largest cost savings are obtained with $L = 11$, that is, with 3 time slices. As for the average CPU ratio, no clear tendency can be observed as it remains relatively stable around 3. Therefore, for the subsequent tests, we use time slices of length $L = 11$ for both BDCA-N and BDCA-N-1 variants.

All instances $I1$ to $I7$ were solved using the BDCA-N and BDCA-N-1 algorithms. Computational results for these tests are given in Table 3. Here again, we report CPU time increase factors, total cost savings and number of schedule savings with respect to the SEQ approach. The results for the BDCA-N variant clearly highlight the efficiency of using neighborhoods. Compared to the BDCA-RC algorithm (see Table 2), the BDCA-N algorithm yields larger total cost savings for all instances. On average, these savings increased from 2.96% to 4.02%, largely exceeding the savings of 3.37% realized by the MPDCA algorithm (see Table 1). The average savings on the number of schedules also improved (from 4.81% to 5.51%), but are similar to those obtained by the MPDCA method (5.54%). What is noticeable is that these gains are achieved without additional computational efforts: the average computational time for the BDCA-N algorithm is almost equal to that of the BDCA-RC algorithm.

The BDCA-N-1 results in Table 3 show that, on average, better quality solutions can be computed when phase 1 is forced after a premature halt in phase 0. In fact, the average total cost savings and the average number of schedules saved increase by 0.74% and 0.34%, respectively, when forcing phase 1. On the other hand, the BDCA-N-1 algorithm requires, on average, approximately 25% more computational time than the BDCA-N algorithm.

Instance	BDCA-N approach			BDCA-N-1 approach		
	CPU ratio	Savings (%)		CPU ratio	Savings (%)	
	BDCA-N/SEQ	Cost	Sched	BDCA-N-1/SEQ	Cost	Sched
<i>I1</i>	1.7	5.74	6.06	1.9	5.72	6.06
<i>I2</i>	2.5	3.60	8.82	3.2	3.28	5.88
<i>I3</i>	3.0	3.07	8.51	5.3	4.46	10.63
<i>I4</i>	1.8	3.42	5.51	2.1	4.02	5.51
<i>I5</i>	6.0	4.09	2.42	6.6	4.97	3.23
<i>I6</i>	3.0	6.75	6.27	4.2	8.78	8.07
<i>I7</i>	3.0	1.50	0.98	3.2	2.14	1.63
Average	3.0	4.02	5.51	3.8	4.76	5.85

Table 3: Results for the BDCA-N and BDCA-N-1 variants

Instance	Value	Value reduction (%)			
	MPDCA	BDCA-DV	BDCA-RC	BDCA-N	BDCA-N-1
<i>I1</i>	719871	-2.43	-1.97	-0.03	0.85
<i>I2</i>	904958	-1.52	-1.72	-1.28	-0.81
<i>I3</i>	1244041	-1.05	-1.10	-0.83	0.10
<i>I4</i>	3381265	-0.21	-0.01	0.65	1.15
<i>I5</i>	4639143	-1.54	-1.09	0.95	1.43
<i>I6</i>	4856988	0.63	0.86	1.94	4.87
<i>I7</i>	6427969	-1.10	-1.10	0.32	0.90
Average		-1.03	-0.87	0.24	1.21

Table 4: Lowest objective values reached by different algorithms

Recall that heuristic rules are used to stop the column generation process (see Section 4.2.4). Therefore, the lowest value that the objective function can reach during the solution process is not necessarily attained at the root node of the search tree. In this respect, it seems interesting to compare the impact of the proposed variants on the lowest value reached. For each instance, Table 4 presents this value for the MPDCA approach of Saddoune *et al.* (2010) and the reductions (increases if negative) yielded by the four proposed solution approaches with respect to these MPDCA values. One can observe that the BDCA-N-1 algorithm allows to obtain the lowest values

for all instances except instance *I2*. On average, the lowest values are computed by the BDCA-N-1, BDCA-N, MPDCA, BDCA-RC, and BDCA-DV algorithms in this order. Remark that the same algorithm ordering would be obtained if the order was determined according to the average total cost savings yielded by the algorithms (see Tables 1, 2, and 3). This supports the assertion that the improvements in the solution quality observed for the BDCA-N and BDCA-N-1 algorithms are due to the use of the neighborhoods that allows reaching lower objective values during the solution process, and not to an unstable variable fixing procedure.

In Saddoune *et al.* (2010), the authors introduced an enhanced two-stage sequential approach that allows pairing concatenation in the crew assignment stage. This partial integration of crew pairing and crew assignment yielded better quality solutions but longer computational times: on average for instances *I1* to *I7*, the total cost was reduced by 1.62%, while the computational time increased by a factor of 2.0 when compared to the traditional sequential approach. When compared to this enhanced sequential approach, the MPDCA approach of Saddoune *et al.* (2010) obtained average total cost savings of 1.76% with an average computational time increase factor of 3.4. This comparison provided a first estimate of the additional savings that can be realized when using complete integration instead of partial integration. Better estimates are now available. Compared to the enhanced sequential approach, the BDCA-N (resp. BDCA-N-1) algorithm produced solutions yielding average total cost savings of 2.43% (resp. 3.20%) for an average time increase factor of 1.4 (resp. 1.8). These results show that a full integration of the crew pairing and crew assignment stages can still be beneficial over a partial integration and that the benefits are achievable without too much additional computational efforts.

6 Conclusion

In this paper, we considered the airline integrated crew scheduling problem and proposed different variants of a combined column generation/bi-dynamic constraint aggregation method for solving it. In particular, we introduced the use of neighborhoods in the bi-dynamic constraint aggregation procedure to increase the possibility of generating complementary columns (that is, columns that can be combined to provoke a decrease in the objective function value) in the column generation process. Computational results on seven instances derived from real-life flight schedules showed that two variants of the proposed solution method can produce better quality solutions than the traditional sequential two-stage solution approach widely used in the industry: average total cost savings of 4.02% and 4.76% were achieved with these variants. With these two algorithms, the

average computational time increases, however, by a factor 3.0 and 3.8, respectively. These time increase factors are much lower than the 6.8 increase factor of the previously developed multi-phase dynamic constraint aggregation method of Saddoune *et al.* (2010), resulting in acceptable computational times for small- and medium-sized instances (up to around 24 hours).

Following this work, several research avenues can be investigated. To further reduce computational times and tackle larger instances, one can explore the use of different neighborhood definitions. Another avenue would be to develop a better way to determine the initial constraint aggregation than starting from a heuristic solution of the crew pairing problem. Finally, one can think about generalizing the proposed model and solution method to produce personalized schedules that would take into account activities preassigned to the crew members such as vacations and training periods as well as crew member preferences.

Acknowledgments: This research was supported by a collaborative R&D grant offered by AD OPT, Division of Kronos and the Natural Sciences and Engineering Research Council of Canada. The authors wish to thank the personnel of AD OPT, Division of Kronos for providing the datasets and the Gencol software library.

References

- Barnhart, C., Johnson E.L., Nemhauser G.L., Savelsbergh M.W.P. and Vance, P.H. (1998). Branch-and-price: column generation for solving huge integer programs. *Operations Research*, **46**, 316–329.
- Barnhart, C. and Shenoi, R.G. (1998). An approximate model and solution approach for the long-haul crew pairing problem. *Transportation Science*, **32**, 221–231.
- Boubaker, K., Desaulniers, G. and Elhallaoui, I., (2010). Bidline scheduling with equity by heuristic dynamic constraint aggregation. *Transportation Research Part B: Methodological*, **44**, 50–61.
- Campbell, K.W., Durfee, R.B. and Hines, G.S., (1997). FedEx generates bid lines using Simulated Annealing. *Interfaces*, **27(2)**, 1–16.
- Christou, I.T., Zakarian, A., Liu, J. and Carter, H., (1999). A two-phase genetic algorithm for large-scale bidline-generation problems at Delta Air Lines. *Interfaces*, **29(5)**, 51–65.
- Desaulniers, G., Desrosiers, J., Dumas Y., Marc, S., Rioux, B., Solomon, M.M. and Soumis, F. (1997). Crew pairing at Air France. *European Journal of Operational Research*, **97**, 245–259.
- Desaulniers, G., Desrosiers, J., Gamache M. and Soumis, F. (1998). Crew scheduling in air transportation. In Crainic, T.G., and Laporte, G., (eds), *Fleet Management and Logistics*, pages 169–185. Kluwer, Boston.

- Desrosiers, J., Dumas, Y., Solomon M.M. and Soumis, F. (1995). Time constrained routing and scheduling. In Ball, M.O., *et al.* (eds), *Network Routing*, Handbooks in Operations Research and Management Science, volume 8, pages 35–139. North-Holland, Amsterdam.
- Elhallaoui, I., Villeneuve, D., Soumis, F. and Desaulniers, G., (2005). Dynamic aggregation of set partitioning constraints in column generation. *Operations Research*, **53**, 632–645.
- Elhallaoui, I., Metrane, A., Soumis, F., and Desaulniers, G. (2008a). Multiphase dynamic constraint aggregation for set partitioning type problems. *Mathematical Programming A*, DOI: 10.1007/s10107-008-0254-5.
- Elhallaoui, I., Desaulniers, G., Metrane, A., and Soumis, F. (2008b). Bi-dynamic constraint aggregation and subproblem reduction. *Computers and Operations Research*, **35**, 1713–1724.
- Gopalakrishnan, B. and Johnson, E.L. (2005). Airline crew scheduling: State-of-the-art. *Annals of Operations Research*, **140**(1), 305–337.
- Irnich, S., and Desaulniers, G. (2005). Shortest path problems with resource constraints. In Desaulniers, G., Desrosiers, J., and Solomon, M.M., (eds), *Column Generation*, pages 33–65. Springer, New York.
- Jarrah, A.I.Z., and Diamond, J.T., (1997). The problem of generating crew bidlines, *Interfaces*, **27**(4), 49–64.
- Klabjan, D. (2005). Large-scale models in the airline industry. In G. Desaulniers, J. Desrosiers, and M.M. Solomon (eds), *Column Generation*, pages 163–195. Springer, New York, NY.
- Klabjan, D., Johnson, E. and Nemhauser, G.L., (2001). Solving large airline crew scheduling problems: random pairing generation and strong branching. *Computational Optimization and Applications*, **20**, 73–91.
- Mercier, A., Cordeau, J.F., and Soumis, F. (2005). A computational study of Benders decomposition for the integrated aircraft routing and crew scheduling problem. *Computers and Operations Research*, **32**(6), 1451–1476.
- Saddoune, M., Desaulniers, G., and Soumis, F. (2009). Aircrew pairing with possible repetitions of the same flight number. Technical report G-2009-76, Les Cahiers du GERAD, HEC Montreal, Montreal, Canada.
- Saddoune, M., Desaulniers, G., Elhallaoui, I., and Soumis, F., (2010). Integrated airline crew pairing and crew assignment by dynamic constraint aggregation. Technical report G-2010-05, Les Cahiers du GERAD, HEC Montreal, Montreal, Canada.
- Vance, P.H., Barnhart, C., Johnson, E.L. and Nemhauser, G.L. (1997). Airline crew scheduling: A new formulation and decomposition algorithm. *Operations Research*, **45**, 188–200.

Weir, J.D. and Johnson, E.L., (2004). A three-Phase approach to solving the bidline problem. *Annals of Operations Research*, **127**, 283–308.

Zeghal F.M, and Minoux, M., (2006). Modeling and solving a crew assignment problem in air transportation. *European Journal of Operational Research*, **175(1)**, 187–209.