Les Cahiers du GERAD ISSN: 0711–2440

A new heuristic branching scheme for the crew pairing problem with base constraints

F. Quesnel, G. Desaulniers, F. Soumis

G–2016–47

June 2016

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

A new heuristic branching scheme for the crew pairing problem with base constraints

Frédéric Quesnel
Guy Desaulniers
François Soumis

GERAD & Polytechnique Montréal, Montréal, Canada H3C 3A7

frederic.quesnel@gerad.ca
guy.desaulniers@gerad.ca
francois.soumis@gerad.ca

June 2016

Les Cahiers du GERAD
G–2016–47

Abstract: Airline crew scheduling is typically performed in two steps : crew pairing followed by crew assignment. The crew pairing problem (CPP) finds a set of pairings (sequences of flights separated by connections or rests starting and ending at the same crew base) that covers a set of flights at minimum cost. The crew assignment problem consists of assigning the crew members to these pairings to create their individual schedules. The main downside of this sequential approach is that the pairings generated in the first step are not all suitable for the crew assignment step, yielding poor-quality solutions. This paper studies an extension of the CPP that includes additional constraints limiting the total worked time at each crew base. This problem, called the CPP with base constraints (CPPBC), is designed to improve the coupling of the two scheduling steps. To solve the CPPBC, we develop four branch-and-price heuristics: three of them rely on known heuristic branching schemes, the other introduces a new branching method, called retrospective branching. This branching scheme is designed to detect and revise poor branching decisions made earlier in the search tree, without backtracking. We tested and compared these four heuristics on real-world datasets. Our results show that the algorithm with retrospective branching yields, most of the times, better-quality solutions than the other tested methods.

Keywords: Airline crew pairing, column generation, branching methods

Résumé : La création d'horaires de personnel aériens est généralement effectuée en deux étapes : la création de rotations d'équipage, suivie par la création d'horaires personnalisés. Le but du problème de rotations d'équipage est de déterminer un ensemble de rotations (séquences de vols séparés par des connections ou repos, débutant et se terminant à une même base), qui couvre un ensemble de vols à coût minimum. Le problème d'horaires personnalisés consiste à assigner ces rotations aux membres d'équipage afin de créer leurs horaires individuels. L'inconvénient principal de cette approche séquentielle est que les rotations générées lors de la première étape sont parfois peu adaptées à la seconde étape. Cela peut résulter en des solutions de faible qualité. Cet article étudie une extension du problème de rotations d'équipage qui inclut des contraintes additionnelles limitant le temps de travail disponible à chaque base. Ce problème, nommé le problème de rotations avec contraintes de bases, a pour but l'amélioration du couplage entre les deux phases. Pour résoudre ce problème, nous développons quatre heuristiques de "branch-and-price". Trois de celles-ci sont basées sur des méthodes de branchement heuristiques connues. Une autre méthode, appelée branchement rétrospectif (retrospective branching), est élaborée dans le but de détecter et d'éliminer les mauvaises décisions de branchement prises plus haut dans l'arbre de branchement, et ceci sans avoir à effectuer de retours en arrière. Nous testons et comparons ces quatre heuristiques sur des données provenant de l'industrie. Nos résultats montrent que le branchement rétrospectif permet de trouver, dans la plupart des cas, des solutions de meilleures qualité qu'avec les autres méthodes testées.

Mots clés : Horaires de personnel aérien, génération de colonnes, méthode de branchement

Acknowledgments: This work was supported by the CRSNG and AD OPT (Division of Kronos) (RDCPJ 477127-14).

1 Introduction

The airline industry is very competitive and, in order to increase their profits, airlines use optimisation techniques in every step of their operation planning. Since crew fees are the second highest source of expense of an airline after fuel, the crew scheduling problem is perhaps the most crucial of these steps. Because of the high complexity and the size of the instances involved, air crew scheduling is usually done in two steps : first crew pairings are built, then crew members are assigned to those pairings. These steps result in two problems called the crew pairing problem (CPP) and the crew assignment problem (CAP). Since employees are qualified to operate on a single aircraft type at a time, each problem can be separated by aircraft type.

Consider a set of flights that must be operated by a certain aircraft type in a given period (e.g., one week or one month). The CPP consists of selecting a set of pairings that covers all flights at minimum cost. A pairing is a sequence of flights starting and ending at the same crew base, spanning one or multiple days and, that can be worked legally by a crew member. Pairings must, in fact, comply with safety regulations and collective agreements in place. Each pairing can be partitioned into duties (sequences of flights corresponding to a day of work), separated by rest periods. Some airlines also allow pairings with deadhead flights (or simply deadheads) where the crew travels as passengers to be relocated. Although costly, the use of deadheads can reduce the overall operation costs.

In the CAP, the pairings obtained in the first step must be assigned to specific crew members to create their monthly schedules. Depending on the airline, different objective functions can be considered in the CAP. For instance, it is possible to balance as much as possible the number of days off and the total working time among the employees or to maximize the satisfaction of the crew members according to preferences that they expressed with respect to specific flights, specific days off, pairing starting times, etc. It is also necessary to take into account the employees' vacations and training schedules, as well as their availability at each crew base.

Solving both problems in a single step would in theory yield better schedules in terms of costs or employee satisfaction, since information on specific employees would be taken into account while building pairings. Approaches integrating both crew planning steps have been proposed in the past (e.g.,Saddoune et al. [1]). However, integrated approaches cannot yet be used in practice for larger fleets because of the large computational times they require. In large airlines, a single aircraft type can typically cover tens of thousands of flights each month and neither the CPP nor the CAP can be solved to optimality. Good quality solutions are instead found using column generation and branching heuristics.

The two-step planning process also poses some challenges. For instance, it is not guaranteed that the pairings obtained by solving the CPP are suitable for the CAP. Indeed, consider the case where the solution to the CPP assigns pairings to a crew base that require more time than the crew members at that base can work. A way to overcome such difficulties is to solve a variant of the CPP that involves constraints related to crew members, such as base constraints limiting the total working time assigned to each base. This favors the creation of pairings that are better suited for the CAP.

The CPP is usually formulated as a set-partitioning model, where variables are associated with feasible pairings. Let F be the set of flights that must be covered and Ω the set of all feasible pairings. With each pairing $p \in \Omega$, we associate a cost c_p and binary parameters a_{fp}, $f \in F$, that indicate whether or not pairing p covers flight f ($a_{fp} = 0$ if the crew assigned to pairing p is deadheading on flight f). Furthermore, let x_p be a binary variable that takes value 1 if pairing p is selected, and 0 otherwise. The set-partitioning model for the CPP is as follows:

$$\min \quad \sum_{p \in \Omega} c_p x_p \tag{1}$$

$$\text{s.t.} \quad \sum_{p \in \Omega} a_{fp} x_p = 1, \qquad \forall f \in \mathcal{F} \tag{2}$$

$$x_p \in \{0,1\}, \qquad \forall p \in \Omega. \tag{3}$$

Objective function (1) minimizes the total pairing costs. Set-partitioning constraints (2) ensure that each flight is covered exactly once by a pairing. Binary requirements (3) restricts the feasible domain of the decision variables.

Adding base constraints to model (1)–(3) typically increases the number of variables taking a fractional value in the solution of its linear relaxation, making it harder to obtain good-quality solutions using the existing solution algorithms. Moreover, one can observe that these algorithms become less efficient as the base constraints become tighter, resulting in poor-quality solutions and larger computational times.

In this paper, we propose a new branching scheme designed to overcome the challenges raised by the CPP with base constraints (CPPBC). This heuristic scheme is an extension of the traditional diving (column fixing) heuristic. It aims at detecting and removing poor branching decisions that were made previously in the search tree, without backtracking. For this reason, we call it the *retrospective branching* (RB) heuristic. We test this new heuristic on weekly instances of the CPP, and compare it with frequently used branching heuristics. Our results show that, for most instances, the RB heuristic yields better quality solutions or smaller computational times than its competitors, especially for the largest instances and when the base constraints are very tight.

The remainder of this paper is structured as follows. Section 2 presents a literature review on the CPP and the related branching heuristics. In Section 3, we provide a detailed definition of the CPPBC that we consider in this paper. Section 4 is devoted to the description of the solution algorithms. We give an overview of the column generation algorithm used to solve linear relaxations, we present briefly the existing branching heuristics that will be used in our computational experiments, and we introduce the RB scheme. Next, computational results are reported in Section 5. Finally, conclusions are drawn in Section 6.

2 Literature review

One of the most straightforward attempt at solving model (1)–(3) relies on an exhaustive enumeration of all pairings (Hu and Johnson [2]). Unfortunately, this approach is not suitable for modern real-life instances, containing typically billions of feasible pairings. A way to circumvent this problem is to solve the CPP with an *a priori* "good" subset of pairings (Klabjan et al. [3], Soykan and Erol [4]). Unfortunately, this approach was shown to yield poor results since it potentially ignores advantageous pairings. Today, most approaches for large-scale instances are based on column generation. Section 2.1 describes the main column-generation-based approaches for the CPP. Different heuristic branching techniques commonly used to solve this problem are reviewed in Section 2.2.

2.1 Column generation

The strength of column generation is to consider implicitly all pairings, while keeping the number of variables at an acceptable level. To achieve this, a restricted master problem (RMP) and one or more subproblems are solved iteratively. The RMP is the linear relaxation of set partitioning formulation (1)–(3) restricted to a limited number of pairing variables (columns). The goal of the subproblems is to find negative reduced cost columns that are added to the RMP. The RMP and the subproblems are solved alternately until no negative reduced cost pairing can be found, guaranteeing that the current optimal solution of the RMP is also optimal for the complete linear relaxation.

Subproblems are usually modeled as shortest path problems with resource constraints in acyclic networks, where a feasible path in a network represents a feasible pairing. There is at least one subproblem per base which ensures that the generated pairings start and end at the same base. The subproblems are usually solved by a labeling algorithm (see Irnich and Desaulniers [5] for further details on the shortest path problem with resource constraints and labeling algorithms). However, Lozano and Medaglia [6] recently proposed a pulse algorithm (an enhanced depth-first search algorithm) to solve the shortest path problem with resource constraints. A parallel version of their algorithm is embedded in a column generation algorithm to obtain optimal linear relaxation solutions to a shift scheduling problem and a transit route design problem.

Two types of networks underlying the subproblems have been used in the literature. In time/space networks, each node corresponds to a time/airport pair, and arcs represent activities (flights, deadheads, rests, etc). This network type has the advantage of being easy to implement, and is suitable when cost functions are relatively simple. Mercier and Soumis [7] rely on this network type to study a variant of the CPP with flexible departure times. Cacchiani and Salazar-González [8] use similar networks to find solutions to the integrated aircraft routing, aircraft assignment and crew pairing problem, for instances containing less than 200 flights. The second type of network, called the duty network, was proposed by Lavoie et al. [9]. In this model, each node represent a duty, and arcs represent rest periods between two consecutive duties. One of the advantages of using this network type is to enable the use of arbitrarily complex duty cost functions, making it more suitable for short-haul problem (Gopalakrishnan and Johnson [10]). This type of network was used by Anbil et al. [11] and Desaulniers et al. [12] to solve in less than one hour CPP instances involving up to 800 flights. The main drawback of using this network type is that it requires the computation of all possible duties beforehand, which can be time consuming or even impossible. Zeren and Özkol [13] use a network structure close to the duty network to find good-quality solutions to CPP instances containing up to 17,000 flight legs.

2.2 Branching algorithms

Generally, the linear relaxation solutions computed by column generation are fractional. Thus, to obtain integer solutions, the column generation algorithm is embedded in a branch-and-bound framework. However, the size of most real-world instances makes it impossible to explore the whole search tree. Consequently, different branching heuristics (see Joncour et al. [14]) have been proposed to derive integer solutions in reasonable computational times. Some authors (e.g., Muter et al. [15], Aydemir-Karadag et al. [16]) apply the so-called restricted master heuristic that consists of solving the linear relaxation of the problem by column generation, before calling a MIP solver to find an integer solution to the current RMP, without generating new columns in the search tree. Unfortunately, this method does not perform well in general because additional columns that complement the branching decisions should often be generated during the branching process to derive good-quality solutions. At the opposite, branch-and-price methods (Barnhart et al. [17], Desaulniers et al. [18]) generate new pairings at each branch-and-bound node, producing better-quality solutions. The remainder of this section reviews such methods, with a focus on branch-and-price heuristics commonly used in the airline industry.

2.2.1 Column fixing

Column fixing algorithms branch on pairing variables without backtracking. In order to impose pairing p on one branch, one has to remove from the RMP all columns containing the flights covered by p, and from the subproblem networks all arcs or nodes representing those flights. It is however difficult, although not impossible (see Villeneuve and Desaulniers [19]), to forbid a specific pairing. In fact, one would have to alter the subproblem networks in a way that prevents the corresponding path to be generated. For this reason, column fixing is mostly used as a heuristic scheme in which only the former type of decisions is applied. It falls in the category of the diving heuristics.

One of the simplest column fixing algorithm is as follows (for instance, see Saddoune et al. [20]). At each node of the search tree, one or several columns taking a fractional-value in the current RMP solution are selected, and their values are set to 1. These columns are selected in decreasing order of their values because columns with larger values have, in general, less impact on the objective function value when they are set to 1 than columns with smaller values. The diving heuristic continues exploring a single branch of the tree in a depth-first fashion until finding an integer linear relaxation solution. An interesting variant of this diving heuristic was proposed by Joncour et al. [14] and showed promising results on cutting stock, bin packing and vehicle routing problems. It allows limited backtracking to explore additional branches which contain solutions that slightly deviate from the first solution found by column fixing.

The main advantage of the diving heuristics is that feasible solutions can be reached relatively fast. However, solution quality can suffer, especially for small instances, where fixing a variable can have a huge impact on the lower bound.

2.2.2 Arc and inter-tasks fixing

Branching on the flow of an arc allows more flexibility than branching on columns because the decisions to force an arc or forbid it in the solution are both compatible with column generation. This feature allows to create two children nodes for each node of the tree. A common practice is to branch only on inter-tasks (also called follow-ons), effectively forcing or forbidding two flights to appear consecutively in the same pairing (Desaulniers et al. [12], Anbil et al. [11], Soykan and Erol [4], Zeren and Özkol [13]). In time-space networks, inter-task branching cannot be directly imposed on the arcs, but require additional resources in the shortest path subproblems with resource constraints (see Irnich and Desaulniers [5]). When studying problems with multiple bases, it is also possible to branch on flight arcs for a particular subproblem. This is equivalent to forcing a flight to be covered by a pairing assigned to a specific base or preventing it. Arc branching decisions can be used in a diving heuristic (Zeren and Özkol [13]), or in an exact branch-and-bound scheme (Soykan and Erol [4]). The latter option is however time-consuming and is generally used in static contexts where no additional columns are generated in the branching tree.

2.2.3 Strong column fixing

In exact branch-and-bound algorithms, strong branching (see, e.g., Klabjan et al. [3]) select a set of candidate pairs of decisions and evaluate these candidates (by solving exactly or heuristically the ensuing linear relaxations) to select the best one. This best pair of decisions is retained and the other are abandoned. In strong column fixing, a similar approach is taken but instead of selecting pairs of candidates, individual candidates are chosen. For the CPP, the candidates correspond to pairing variables (with the highest fractional values) that are forced to take value 1. Each candidate decision is evaluated by solving the linear relaxation that results from this decision. The retained candidate is the one yielding the best lower bound. Strong branching has the advantage of exploring different column fixing options at each node without incurring an exponential growth of the search tree. However, the performance of this method can be poor when fixing different columns has the same effect on the lower bound, since the additional work required to solve multiple relaxations does not provide new information.

3 Problem definition

This paper addresses the CPPBC. As for the CPP, the goal of the CPPBC is to find a set of feasible pairings that cover every flight exactly once at minimum cost. In addition, the CPPBC includes additional constraints limiting the total working time assigned to each base. In this section, we give a detailed definition of the CPPBC that we consider and we provide a mathematical formulation for the CPPBC.

In practice, the rules governing the feasibility of a pairing are very complex and may vary greatly from one airline to another. Our study considers a core subset of rules, namely, those commonly used in the literature. As mentioned in the introduction, a pairing starts and ends at a crew base. It can be seen as a sequence of duties interspersed with rest periods and each duty corresponds to a sequence of flights separated by connections. On a flight, a crew may be active or in deadhead. We define the time worked in a duty as the total active flying time, plus half of the deadhead time. A crew can work at most 8 hours in a duty and must be paid at least 4 hours per day, whether the crew members actually work for that duration or not. The length (span) of a duty cannot exceed 12 hours and that of a pairing 4 days. A duty can contain at most 4 flights (active or deadhead) and a pairing at most 5 duties. Finally, two consecutive flights in a duty must be separated by a connection of at least 30 minutes and two consecutive duties in a pairing must be separated by a rest period of at least 9.5 hours. Note that all these values can vary from one airline to another and that the minimum connection time could be airport-dependent.

The cost c_p of a pairing $p \in \Omega$ is computed according to a realistic non-convex function that involves three components: one related to the total paid time, one to the deadheads, and another to short connections and rests. Below, we describe these three components.

Let δ_f be the duration of flight f, D_p the set of duties in pairing p, $\mathcal{F}_d$ the set of active flights in duty d, and H_d the set of deadheads in duty d. The total paid time in pairing p is given by:

$$T_p = \max\left\{\frac{\delta_p}{4}, \sum_{d\in D_p} \max\left\{4, \sum_{f\in\mathcal{F}_d}\delta_f + \frac{1}{2}\sum_{f\in H_d}\delta_f\right\}\right\} \tag{4}$$

It consists of the maximum between a quarter of the total duration δ_p of the pairing and the sum of the actual worked time in each duty, taking into account the 4-hour minimum paid time per duty.

Each deadhead flight $f \in H_d$, $d \in D_p$, incurs a fixed cost γ^{DH}, plus a variable cost proportional to the duration of f, at a rate of λ^{DH} per minute. For our tests, we used $\gamma^{DH} = 400$ and $\lambda^{DH} = \frac{5}{6}$.

To favor crew schedules that are robust to delays, short connections and short rests are penalized as follows. Let $\underline{t}^C$ and $\underline{t}^R$ be the minimum acceptable times for a connection and a rest, respectively. Note that any idle period between two consecutive flights in a pairing is considered a connection if it lasts less than $\underline{t}^R$ and a rest otherwise. Let $\bar{t}^C$ and $\bar{t}^R$ be the target times for a connection and a rest, respectively. These target times are such that any connection or rest lasting less than their respective target is considered short (non-robust) and must be penalized at a rate of ϵ^C per minute for a connection and of ϵ^R per minute for a rest. We assume that $\underline{t}^R > \bar{t}^C$. Let W_p be the set of connections and rests in pairing p and denote by δ_w the duration of idle period $w \in W_p$ in minutes. Then, the function $\phi(\delta_w)$ penalizing a short connection or a short rest $w \in W_p$ with respect to its duration δ_w is defined by:

$$\phi(\delta_w) = \begin{cases} \epsilon^C(\bar{t}^C - \delta_w) & \text{if } \underline{t}^C \le \delta_w < \bar{t}^C \\ \epsilon^R(\bar{t}^R - \delta_w) & \text{if } \underline{t}^R \le \delta_w < \bar{t}^R \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

For our tests, we used $\underline{t}^C = 30$, $\bar{t}^C = 90$, $\underline{t}^R = 570$, $\bar{t}^R = 690$, $\epsilon^C = 6$, and $\epsilon^R = \frac{25}{6}$, where the times are in minutes.

Given these three components, the cost c_p of a pairing $p \in \Omega$ is given by:

$$c_p = T_P + \sum_{f\in H_p}\left(\gamma^{DH} + \lambda^{DH}\delta_f\right) + \sum_{w\in W_p}\phi(\delta_w), \tag{6}$$

where $H_p = \bigcup_{d\in D_p} H_d$ denotes the set of deadhead flights in p.

The key feature of the CPPBC is to include base constraints that aim at sharing the worked time between the crew bases proportionally to the number of employees in each base. Since it is difficult in practice to determine precisely the amount of time required to operate a flight schedule, modeling base constraints as hard constraints is not appropriate. Instead, the excessive allocation of worked time at each base is discouraged through penalties in the objective function. Let $\mathcal{B}$ be the set of crew bases. For base $b \in \mathcal{B}$, the penalty is defined by a piecewise linear convex function (see Figure 1) whose set of pieces is denoted S_b and a non-strict maximum worked time M_b. The lengths U_{sb} and slopes ρ_{sb} of the segments $s \in S_b$ are set relative to M_b: no penalty is incurred on the first segment, a small unit penalty must be paid on the next few segments until reaching M_b, then the unit penalty increases rapidly with the segments after M_b. Our implementation of the CPPBC uses 12 segments for each base constraint.

To formulate the CPPBC, we adapt model (1)–(3) as follows. Let $\Omega_b \subseteq \Omega$ be the subset of pairings assigned to base $b \in \mathcal{B}$. For each base $b \in \mathcal{B}$ and each segment $s \in S_b$, define a non-negative variable y_{sb} whose value indicates the amount of worked time at base b that is penalized on segment s. The proposed model for the CPPBC is:

$$\min \quad \sum_{p\in\Omega} c_p x_p + \sum_{b\in\mathcal{B}}\sum_{s\in S_b}\rho_{sb}y_{sb} \tag{7}$$

$$\text{s.t.} \quad \sum_{p\in\Omega} a_{fp}x_p = 1, \qquad \forall f \in \mathcal{F} \tag{8}$$

$$\sum_{p\in\Omega_b} T_p x_p = \sum_{s\in S} y_{sb}, \qquad \forall b \in \mathcal{B} \tag{9}$$

$$0 \leq y_{sb} \leq U_{sb}, \qquad \forall b \in \mathcal{B}, s \in S_b \tag{10}$$

$$x_p \in \{0,1\}, \qquad \forall p \in \Omega \tag{11}$$

The objective function (7) seeks to minimize the total pairing costs plus the sum of the base constraint penalties. Constraints (8) ensure that each flight is actively covered by exactly one pairing. Constraints (9)–(10) define the segments of the piecewise linear penalties and compute the values of the variables y_{sb}. Note that the convexity of the piecewise linear penalty functions ensures that, in an optimal solution, a variable y_{sb} can take a positive value only if $y_{s'b} = U_{s'b}$ for any segment s' preceding segment s in S_b, $b \in \mathcal{B}$. Finally, the pairing variables x_p, $p \in \Omega$ are subject to the binary requirements (11).

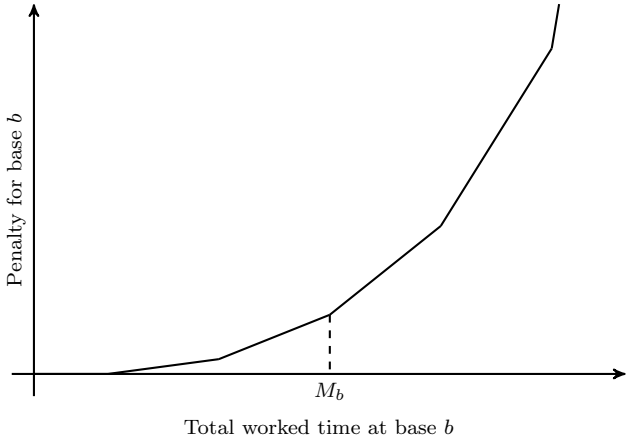


Figure 1: Piecewise linear penalty for the base b constraint

4 Solution algorithms

To solve model (7)–(11), we consider four branch-and-price heuristics: three of them rely on existing branching methods and the other uses a new heuristic branching called the RB. Section 4.1 presents the column generation algorithm used to solve the linear relaxations in all branch-and-price heuristics. Section 4.2 describes the three traditional branching methods whereas Section 4.3 introduces the RB heuristic.

4.1 Column generation

In a column generation framework, an RMP and several subproblems are solved iteratively. The RMP consists of the linear relaxation of model (7)–(11), with Ω replaced by $\Omega' \subseteq \Omega$, a subset of all feasible pairings. Set Ω' is augmented at each iteration with new negative reduced cost pairings found by solving the subproblems.

To obtain an optimal solution to the linear relaxation, one must solve iteratively the RMP and the subproblems until no negative reduced cost pairing is found. However, column generation is known to converge slowly when getting close to the optimal value, i.e., it suffers from a tailing effect. Indeed, close to optimality, the negative reduced cost pairings added to Ω' have little or no effect on the optimal value of the RMP. To avoid this situation, we prematurely stop column generation when less than 0.1% improvement on the RMP optimal value has been achieved in the last 5 iterations. The main drawback of this accelerating strategy is the lost of the optimality certificate that column generation usually provides. However, this has little consequence in the context of this paper because we develop branch-and-price heuristics that would not be exact even if linear relaxations were solved to optimality.

The subproblems are shortest path problems with resource constraints defined on acyclic networks. Their goal is to identify negative reduced cost pairing variables. Let $\pi_f^{(8)}$, $f \in \mathcal{F}$, and $\pi_b^{(9)}$, $b \in \mathcal{B}$, be the dual

variables associated with contraints (8) and (9), respectively. The reduced cost of variable x_p for a pairing p assigned to base $b \in \mathcal{B}$ is given by

$$\begin{aligned}
\bar{c}_p &= c_p - \sum_{f\in\mathcal{F}} \pi_f^{(8)} a_{fp} - \pi_b^{(9)} T_p \\
&= T_P + \sum_{f\in H_p} \left(\gamma^{DH} + \lambda^{DH}\delta_f\right) + \sum_{w\in W_p} \phi(\delta_w) - \sum_{f\in\mathcal{F}} \pi_f^{(8)} a_{fp} - \pi_b^{(9)} T_p \\
&= (1-\pi_b^{(9)}) \max\left\{\frac{\delta_p}{4}, \sum_{d\in D_p} \max\left\{4, \sum_{f\in\mathcal{F}_d}\delta_f + \frac{1}{2}\sum_{f\in H_d}\delta_f\right\}\right\} + K \\
&= \max\left\{(1-\pi_b^{(9)})\frac{\delta_p}{4} + K, (1-\pi_b^{(9)}) \sum_{d\in D_p} \max\left\{4, \sum_{f\in\mathcal{F}_d}\delta_f + \frac{1}{2}\sum_{f\in H_d}\delta_f\right\} + K\right\}, \qquad (12)
\end{aligned}$$

where $K = \sum_{f\in H_p}\left(\gamma^{DH} + \lambda^{DH}\delta_f\right) + \sum_{w\in W_p}\phi(\delta_w) - \sum_{f\in\mathcal{F}}\pi_f^{(8)} a_{fp}$. For the last equality, we assume that $1-\pi_b^{(9)} \geq 0$. If this is not the case, the first maximum must be replaced by a minimum.

There is one subproblem for each base and each day during which a pairing can start. Let $G = (N, A)$ be the network of a given subproblem, where N and A are its node and arc sets, respectively. Such a network is illustrated in Figure 2. Set N contains one *source* node; one *sink* node for each day that a pairing can end if it starts on the day associated with the subproblem; and three nodes for each flight, namely, a *departure* node, an *arrival* node, and a *waiting* node. Arc set A contains the following arcs. For each flight starting the same day that the pairing begins, a *start of pairing* arc links the source node to the departure of the flight and an *end of pairing* arc links the arrival node of the flight to the sink node that corresponds to the day that the flight ends. Each flight is represented by a *flight* arc and a *deadhead* arc linking its departure node to its arrival node. The arrival node of each flight f is linked by outbound *short connection* arcs to the departure nodes of all flights whose departure airport is the same as the arrival airport of f and whose departure time is early enough for the connection to be considered short. This arrival node is also linked by a *long connection* arc to the waiting node of the first flight whose departure airport is the same as the arrival airport of f and for which the connection time is too long to be considered short. Similarly, the arrival node of each flight is connected by *short rest* arcs to the departure nodes of all flights whose departure is late enough to yield a rest, but early enough for the rest to be considered short. *Long rest* arcs also link the arrival node of each flight to the waiting node of the earliest flight whose departure time is too late for a short rest. There is an *empty* arc between each waiting node and its corresponding departure node. The waiting nodes are ordered chronologically at each airport, and each waiting node is connected to its successor by a *waiting* arc.

The cost structure and feasibility constraints of the pairings are modeled by resource constraints on the network. A resource is a quantity containing information about a partial path in the network (for example, the total elapsed time). Resources are consumed when traversing arcs in the network. At each node, the value of every constrained resource must fall within a resource interval, called a *resource window*. The set of resource values associated with a partial path is stored in a vector called a *label*.

Our model uses the following six resources to constrain the pairings and compute the cost function. The *number of flights* resource limits the number of flights that a crew member can take in a duty. Another resource, called *number of duties*, limits the number of duties in a pairing. The *duty length* and *duty worked time* resources are used to limit the total length and total worked time in each duty, respectively. Two resources are required to compute the cost of a pairing since it depends on the maximum between the two terms in (12). Consequently, we use one resource to compute the *reduced cost according to the length of the pairing* (the first term in (12)), and one for the *reduced cost according to the worked time* (the second term in (12)). When reaching a sink node, the real reduced cost of a pairing is set as the largest value taken by these two resources. We denote by $\mathcal{R}$ the set of resources and by $[l_i^r, u_i^r]$, $i \in N$, $r \in \mathcal{R}$, the resource window at node $i \in N$ for resource $r \in \mathcal{R}$. For example, for the number of flights resource, the resource window is $[0, 4]$ at every node $i \in N$. The resource windows are not constraining for the last two resources.

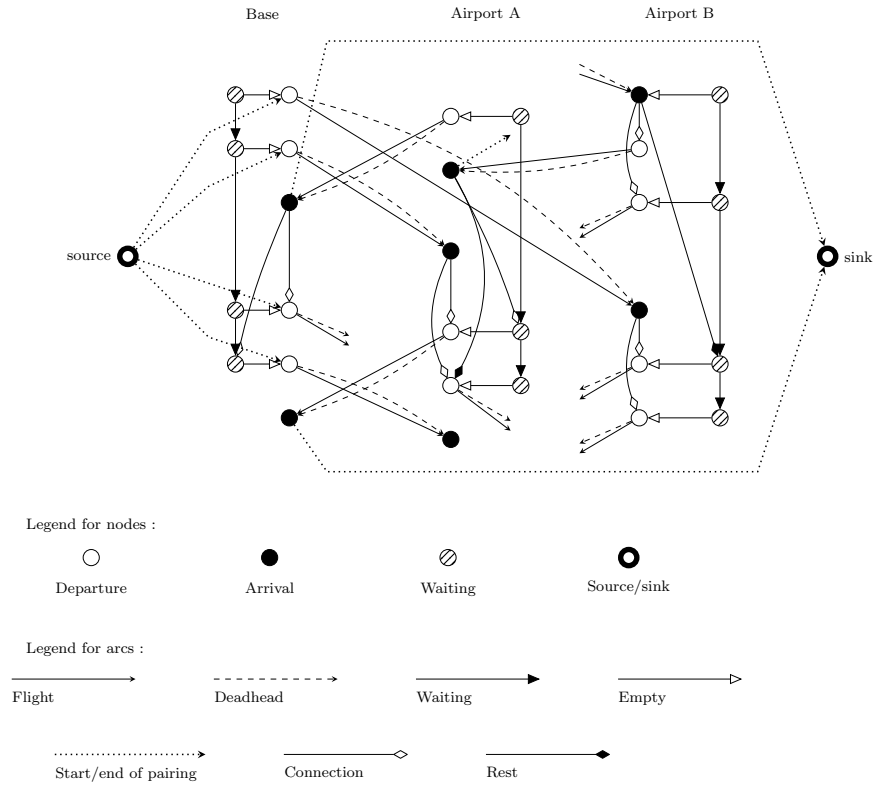


Figure 2: Network structure for the subproblems

The subproblems are solved by a labeling algorithm. Labels are extended from the source node to the sink nodes in order to find one or more negative reduced cost pairings. When the partial path represented by a label $L_i = (T_i^r)_{r\in\mathcal{R}}$ is extended along an arc $(i,j) \in A$ where T_i^r is the value of resource r, resource values are updated using *resource extension functions* $g_{ij}^r(L_i)$, $r \in \mathcal{R}$ to yield a new label $L_j = (T_j^r)_{r\in\mathcal{R}} = (\max\{l_j^r, g_{ij}^r(L_i)\})_{r\in\mathcal{R}}$ associated with node j. For example, consider the case where r_1 is the number of flights resource and $T_i^{r_1} = 2$ at a given node i (i.e., the current partial duty contains two flights). If (i,j) is a flight or a deadhead arc, then $g_{ij}^{r_1}(L_i) = T_i^{r_1} + 1 = 3$ and $T_j^{r_1} = \max\{0,3\} = 3$. On the other hand, if (i,j) is a short or long rest arc, then $g_{ij}^{r_1}(L_i) = T_i^{r_1} - 4 = -2$ and $T_j^{r_1} = \max\{0,-2\} = 0$, which resets the resource value to 0 before starting a new duty. The partial path associated with the new label L_j is deemed feasible only if $T_j^r \leq u_j^r$. If this is not the case, the label is deleted. Finally, to avoid enumerating all feasible paths, labels are eliminated using a dominance rule. In our case, because all resource extension functions are non-decreasing, we use the standard dominance rule : a label $L_1 = (T_1^r)_{r\in\mathcal{R}}$ dominates a label $L_2 = (T_2^r)_{r\in\mathcal{R}}$ (which can be discarded) if $T_1^r \leq T_2^r$ for all $r \in \mathcal{R}$. If both labels dominate each other, then one of them must be kept. For further details on labeling algorithms for solving shortest path problems with resource constraints, the reader is referred to Irnich and Desaulniers [5].

4.2 Traditional branching methods

Several branching methods can be used to derive integer solutions in a branch-and-price framework. In this section, we describe the three traditional branching methods considered in this paper.

4.2.1 Diving column fixing

As described in Section 2.2.1, the diving column fixing (DCF) heuristic consists of rounding up pairing variables at each node without backtracking. At a node, the columns to fix are selected in decreasing order of their values. Their number is restricted by the following rules.

1. If no columns take a fractional value greater than or equal to a predefined threshold $\tau^C \in]0,1[$, the column with the largest value is the only one selected. Otherwise, the selected columns must have a fractional value greater than or equal to τ^C. A large value of τ^C will strongly restrict the number of variables fixed at once, slowing down the solution process. On the other hand, a small value can allow many columns with relatively small values to be fixed simultaneously, resulting sometimes in poor-quality solutions.
2. Defining $1 - x_p$ as the complementary value of pairing variable x_p, the sum of the complementary values of the fixed columns must not exceed a maximum value $\xi^C \geq (1 - \tau^C)$. This parameter is used to control the number of variables fixed that have values close to τ^C.
3. A maximum of μ^C columns can be fixed at once.

4.2.2 Strong column fixing

Strong column fixing (SCF) combines strong branching on the columns and column fixing as follows. At each node, column fixing is applied if there exists at least one column with a fractional value greater than or equal to a predefined threshold $\tau^C \in]0,1[$. Otherwise, strong branching is applied and a maximum of μ^S candidate columns are evaluated before deciding which decision to retain. All candidate columns must have a value over another positive threshold $\tau^S < \tau^C$. If there are no columns with a value greater than or equal to τ^s, the column with the largest fractional value is fixed.

4.2.3 Column fixing and arc branching

Arc branching differs from the other branching methods presented above because two children nodes are created at each node. However, this method can not be used alone in practice for the CPPBC because the resulting search tree would be too large, yielding very large computational times. In consequence, we combine column fixing with arc branching to give the CFAB heuristic. At each node, columns are fixed according to the diving column fixing heuristic if at least one column with a fractional value above τ^C can be fixed. Otherwise, branching is performed on the flow on an arc in one of the subproblem networks. In this case, we branch on an arc whose flow has the maximum distance to its closest integer.

4.3 Retrospective branching

RB aims at circumventing the weaknesses of the traditional branching heuristics, while maintaining reasonable computational times. In particular, it tries to avoid backtracking by detecting and revising poor decisions made previously in the search tree. These decisions are selected from a list of *risky* decisions that is maintained during the branching process. When the relative gap q_i between the values z_i and z_0 of the solutions computed at a node i in the search tree and at the root node (i.e., $q_i = (z_i - z_0)/z_0$) exceeds an estimate E^R of the relative gap for a good integer solution, the algorithm ejects one risky decision from the current solution without backtracking. This ejection is performed via a constraint that is updated dynamically during branching.

The RB heuristic uses the following additional parameters: τ^C, μ^C, and ξ^C as defined in the column fixing method; $\Delta^R > 0$ is an increment of the estimated relative gap; and $\nu^R > 0$ is the maximum number of risky column decisions that can be revised. It also relies on the following sets at node i of the search tree: X_i is the set of variables fixed to 1 and I_i is the set of inherited constraints to be defined later.

The RB heuristic is outlined in Algorithm 1 (together with the functions given in Algorithms 2, 3, and 4). The main thread of this heuristic is as follows. It starts as the DCF heuristic using parameters τ^C, μ^C, and ξ^C to select columns to fix at each node of the search tree. At a node i, when there are no columns with a fractional value greater than or equal to τ^C, then the column with the largest fractional value is selected. However, instead of fixing this column to 1, it is declared as a risky column. Indeed, if it was fixed to 1, it would have a high chance of causing a relatively large increase of the objective function value later in the search tree, given its low value. The risky column is added to the set of risky columns associated with the child node of i. Risky columns are not fixed directly but they are all imposed by a single constraint in the master problem which evolves during the search, as new risky columns are identified. Furthermore, when the

linear relaxation at a node i yields a relative gap q_i that exceeds E^R, then a backtrack occurs and a sibling node is created that imposes to use one fewer risky column without identifying it.

Let Y_i be the set of risky columns at a node i and $n_i \leq \nu^R$ the number of risky column decisions to revise at this node. The constraint associated with the risky columns at node i is:

$$\sum_{p \in Y_i} x_p = |Y_i| - n_i. \tag{13}$$

When $n_i = 0$, this constraint is equivalent to fixing all risky columns to 1. Otherwise, exactly n_i risky columns cannot be part of a feasible solution.

Algorithm 1 Retrospective branching heuristic

1: Set values of parameters E^R, Δ^R, and ν^R
2: Set values of column fixing parameters τ^C, μ^C and ξ^C
3: $s^* := \emptyset$, $z^* := \infty$, $X_0 := \emptyset$, $I_0 := \emptyset$, $Y_0 := \emptyset$, $n_0 := 0$
4: SOLVE_NODE(0)

Algorithm 2 Function SOLVE_NODE(node index i)

5: Solve P_i, the linear relaxation of (7)-(11), (13), constraints (14) in I_i, and fixed variables in F_i
6: **if** P_i is feasible **then**
7: Denote by s_i and z_i its computed solution and value, respectively
8: Compute q_i
9: **else**
10: $q_i := \infty$
11: **end if**
12: **if** an integer solution with a better value than z^* was found while solving P_i **then**
13: Update s^* and z^* with the best found solution
14: **end if**
15: Prune any node j such that $z_j \geq z^*$
16: **if** $\big(z_i \geq z^*$ **or** $(q_i \leq E^R$ **and** s_i is integer**)** **or** $(q_i > E^R$ **and** $n_i = \nu^R)\big)$ **then**
17: **if** $\exists$ at least one unpruned node j **then**
18: Select among them a node j with the least gap q_j
19: CREATE_CHILD($i+1$,j)
20: **if** $q_j > E^R$ **then**
21: $E^R := q_j + \Delta^R$
22: **end if**
23: **else**
24: STOP and RETURN s^* and z^*
25: **end if**
26: **else if** $\big(q_i \leq E^R$ **or** $Y_i = \emptyset\big)$ **then**
27: CREATE_CHILD($i+1$,i)
28: **else**
29: CREATE_SIBLING($i+1$,i)
30: **end if**
31: SOLVE_NODE(i+1)

After solving the linear relaxation at a node i (Step 7)[1] and performing simple operations (Steps 8–15), the following three cases can occur.

1. If $z_i \geq z^*$ or $(q_i \leq E^R$ and s_i is integer) or $(q_i > E^R$ and $n_i = \nu^R)$ (Step 16), we do not create a child node to node i. In the first two cases, node i is pruned; in the last case, node i seems unpromising according to the gap estimate and we may decide to create a child node later. Consequently, if there

[1] The steps in Algorithms 1–4 are numbered consecutively.

exist unpruned nodes (Step 17), we select the best one (it might be node i), create a child node $i+1$ to this selected node j, and update the gap estimate if $q_j > E^R$. Otherwise, there are no more unpruned nodes and the algorithm stops.

2. If the first case does not occur and $q_i \leq E^R$ (Step 24), then there is still hope to find a good solution in this branch and a child node $i+1$ to node i is created. On the other hand, if $q_i > E^R$ but no risky column decisions have yet been made ($Y_i = \emptyset$), then we have no choice to also create a child node $i+1$ to node i.
3. If the first two cases do not occur, then $Y_i \neq \emptyset$, $q_i > E^R$ and $n_i < \nu^R$ (Step 26). We suspect that the objective value increased too much because of the number of risky column decisions made. In consequence, we stop exploring this branch (possibly temporarily) and create a sibling node $i+1$ to node i. In this sibling node, we increase by 1 the number of risky column decisions to revise.

Once node $i+1$ is created, the algorithm calls again the function SOLVE_NODE() to solve this new node in a recursive fashion.

When creating a child node with function CREATE_CHILD(), the algorithm fixes columns in the following priority order: non-risky columns with values greater than or equal to τ^C as in the DFC heuristic (Steps 27–29); one non-risky column with the highest value (Steps 30–32); and one risky column with the largest fractional value.

The RB heuristic is enhanced with two acceleration strategies. First, when creating a sibling node $i+1$ to a node i, node $i+1$ inherits a constraint derived from the risky column set at node i which prevents solution s_i to reappear in the subtree rooted at node $i+1$. This *inherited* constraint is given by:

$$\sum_{p \in Y_i} x_p \leq |Y_i| - n_i + 1 \tag{14}$$

and enforces that at least one additional risky column in set Y_i be discarded. This constraint is not useful at node $i+1$. However, it will be transmitted to its descendant nodes where it could become useful if additional risky columns are added to the set of risky columns. We denote by I_i the set of inherited constraints at node i.

The second acceleration strategy is a feasibility check of constraints (13) that is performed after adding a new risky column in Step 32 in function CREATE_CHILD(). Given that, at a node i with $n_i \geq 1$, the set Y_i of risky columns can contain columns associated with pairings that cover the same flight, it may happen that, at this node, the maximum number of risky columns that can take value 1 simultaneously in an integer feasible solution is less than or equal to $|Y_i| - n_i$, the number of these risky columns that must be set to 1. However, this number $|Y_i| - n_i$ may be reached by a fractional solution, and a subtree rooted at node i, possibly containing many nodes, may have to be explored before concluding that no integer feasible solution can be found in this subtree. To avoid exploring these nodes, we compute the maximum number of risky columns in Y_i that can take value 1 simultaneously by solving the following small integer program:

$$\max \quad \sum_{p \in Y_i} x_p \tag{15}$$

Algorithm 3 Function CREATE_CHILD(created node index i, parent node index j)

32: **if** s_j contains at least one variable x_p such that $\tau^C \leq x_p < 1$ and $p \notin Y_j$ **then**
33: Select an index set U of the variables to fix using the column fixing heuristic
34: $X_i := X_j \cup U$, $I_i := I_j$, $Y_i := Y_j$, $n_i = n_j$
35: **else if** s_j contains at least one variable x_p such that $0 < x_p < \tau^C$ and $p \notin Y_j$ **then**
36: Select among them one variable x_u with the largest value
37: $X_i := X_j$, $I_i := I_j$, $Y_i := Y_j \cup \{u\}$, $n_i := n_j$
38: **else**
39: Select a variable x_u with the largest fractional value
40: $X_i := X_j \cup \{u\}$, $I_i := I_j$, $Y_i := Y_j \setminus \{u\}$, $n_i = n_j$
41: **end if**

Algorithm 4 Function CREATE_SIBLING(created node index i, sibling node index j)

42: $X_i := X_j$, $I_i := I_j \cup \{ \sum_{p \in Y_j} x_p \leq |Y_j| - n_j + 1\}$, $Y_i := Y_j$, $n_i := n_j + 1$

$$\text{s.t.} \quad \sum_{p \in Y_i} a_{fp} x_p \leq 1, \qquad \forall f \in \mathcal{F}(Y_i) \tag{16}$$

$$x_p \in \{0,1\}, \qquad \forall p \in Y_i, \tag{17}$$

where $\mathcal{F}(Y_i)$ is the set of all flights covered by columns in Y_i. If the optimal value of model (15)-(17) is not greater than or equal to $|Y_i| - n_i$, node i is pruned and the algorithm continues like in Steps 15–21.

To conclude this section, let us mention that the addition of constraints (13) and (14) to the RMP introduce new dual variables that impact the reduced cost of the columns in Y_i at node i. Normally, the subproblems should be modified to take into account these dual variables and to ensure that no columns in the risky column set Y_i are generated again. However, this is very difficult to implement since the dual variables associated with constraints (13) and (14) are related to specific pairings. In consequence, we adopted a heuristic strategy that consists of inspecting every negative reduced cost columns found by the subproblems and adding to the RMP only those that are not identical to a column already in Y_i. Note that this approach could in theory prevent negative reduced cost columns from being generated because of the dominance rule that is not adjusted for this case. However, our computational tests indicate that this behavior is marginal because columns in the risky column set are infrequently regenerated.

5 Computational results

In this section, we report the results of our computational experiments that we conducted to compare the performance of the branching heuristics described in Sections 4.2 and 4.3, namely, DCF, SCF, CFAB, and RB. First, we present the instances used for our tests in Section 5.1 and the parameter setting used for each branching heuristic in Section 5.2. Section 5.3 reports our main comparative results. Finally, a sensitivity analysis on the parameter values used for the new RB heuristic is performed in Section 5.4.

5.1 Instance description

Our instances are derived from the 7 datasets of Kasirzadeh et al. [21] which vary in size and originate from a major North-American airline. Each dataset contains a one-month flight schedule, as well as a list of airports and bases. Because we focus on a weekly problem which may occur when applying a rolling-horizon approach for solving a monthly problem, only the data related to the second week (days 6 to 12) are considered in our instances. Data related to the first 7 days are, however, used to compute a partial solution spanning days 1 to 5. This solution is computed by the algorithm of Saddoune et al. [20] and may contain pairings that are not completed at the end of day 5. In our model, we impose the completion of these incomplete pairings by adding for each of them a constraint in the master problem and a subpath representing it in the first subproblem associated with its base.

The characteristics of each instance (numbers of flights, airports, and bases) are displayed in Table 1. The instances are divided in two groups according to their size. Instances 1, 2 and 3 contain between 271 and 479 flights and are considered small. They are however of importance to this study because smaller instances tend to yield larger integrality gaps. Instances 4, 5, 6 and 7 are much larger, with sizes ranging from 1463 to 1987 flights.

For each instance, we study different distribution scenarios of the non-strict maximum total worked time M_b for each base $b \in \mathcal{B}$. The number of scenarios tested for each instance is also reported in Table 1. Less scenarios are considered for the larger instances because they require more computational time. In order to create realistic scenarios for each instance, we first observed how pairings would distribute themselves when the instance is solved without base constraints. We then created the scenarios by using similar proportions,

gradually decreasing the maximum worked time. Typically, half of the available worked time is allocated to one base and the other half is divided unevenly between the other two bases.

Table 1: Instance characteristics

Instance	Group	Flights	Airports	Bases	Scenarios
1	small	271	23	3	32
2	small	479	38	3	24
3	small	395	32	3	21
4	large	1463	31	3	12
5	large	1471	49	3	12
6	large	1987	51	3	12
7	large	1414	46	3	12

5.2 Parameter settings

To compare the branching heuristic fairly, we determine the best parameter setting for each of them separately by testing a large number of settings over multiple instances and scenarios. Because of the significant difference between the sizes of the small and the large instances, the parameter setting was also adjusted according to the instance group. In all cases, the parameter setting selected is the one offering the best compromise between a small average integrality gap and a small average computational time, with an emphasis on the former criterion. The parameter setting found for each branching method and each instance group is specified in Table 2. Note that, for the large instances, our preliminary tests showed that CFAB yields too large computational times to be applicable.

Table 2: Parameter setting for each branching heuristic and each instance group

		Small instances	Large instances
DCF	τ^C	0.8	0.7
	μ^C	3	∞
	ξ^C	1.0	1.5
SCF	τ^C	0.75	0.7
	μ^C	3	∞
	ξ^C	∞	1.5
	τ^S	0.7	0.6
	μ^S	3	3
CFAB	τ^C	0.6	N/A
	μ^C	∞	N/A
	ξ^C	1.5	N/A
RB	τ^C	0.9	0.7
	μ^C	2	∞
	ξ^C	0.4	1.5
	E^R	0.3	0.2
	Δ^R	0.3	0.1
	ν^R	2	1

5.3 Comparative results

This section presents computational results that are used to compare the performance of the four branching heuristics. These results are provided in Tables 3–9, one table per instance. In these tables, each row, except a few ones, corresponds to a distinct distribution scenario of the maximum worked time per base. A scenario is represented by a triplet given the targeted worked time for each base. For example, 2500/1650/845 indicates a scenario in which bases 1, 2 and 3 have maximum worked times of 2500, 1650 and 845 hours, respectively. An unlimited scenario is equivalent to the CPP without base constraints. In each table, the scenarios are presented in decreasing order of the total maximum worked time. For each scenario and each tested heuristic, we report the computational time in seconds and the gap in percentage between the value of the best integer solution found and the objective value reached at the root node of the search tree. Given that the linear

relaxation solutions computed by all heuristics are identical, these gaps can be used to compare the quality of the solutions obtained by these heuristics. In Tables 6–9, we also specify for each scenario and each heuristic other than DCF the relative time and gap differences with respect to the time and gap obtained by DCF. Finally, additional rows provide averages over some scenarios.

All experiments were conducted on a Linux computer equipped with an Intel Core i7-1770 CPU clocked at 3.40 GHz, using a single core and a single thread. The algorithms were coded in C and C++ using the commercial Gencol library, version 4.5, which is specialized for the implementation of branch-and-price algorithms. All RMPs are solved using the primal simplex algorithm of Cplex 12.4.0.0.

5.3.1 Small instances

We start by analyzing the results obtained for the small instances which are presented in Tables 3–5. Scenarios for instances 1 and 2 are separated in three categories according to the thigthness of their base constraints. Our analysis focuses mainly on instances 1 and 2, since the results for instance 3 are quite similar for all branching heuristics, except for the CFAB method. Instance 3 is easier to solve because only a few variables are fractional at the root node of the search tree. In fact, integer solutions are often found at this node. For instances 1 and 2, we observe that, while DCF is able to find solutions with gaps below 1% for most scenarios in which the maximum worked times are not tight, it struggles to find good-quality solutions for more constrained scenarios. This is explained by the fact that, when base constraints are restrictive, the root node solutions contain a larger number of fractional-valued pairing variables in order to evenly split the worked time between the bases. This causes an increase in the number of branching nodes required to obtain a good integer solution. Since more columns are fixed, there is a higher risk of fixing low-value columns that increase drastically the objective value, resulting in the end in poor-quality solutions. To support this hypothesis, we computed the number of nodes in the search tree and the number of fractional-valued variables in the linear relaxation solution when DCF is applied to solve instance 2 for several scenarios. Figure 3 presents the results, plotted against the total maximum worked time. We observe that both the number of branching nodes and the number of fractional-valued variables in the linear relaxation solution decrease as the total maximum worked time increases.

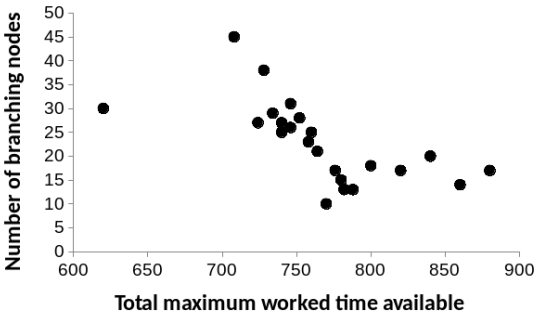
The SCF heuristic produces results similar to those obtained by DCF. On average, SCF finds solutions with slightly smaller gaps than DCF, in similar computational times. However, we observe that the SCF heuristic finds solutions with significantly smaller gaps in highly constrained scenarios.

The first observation we make about the CFAB heuristic is that, while solutions are found in reasonable times for most scenarios, a few of them require much larger computational times. This behavior is undesirable in practice because the airline planners expect relatively stable computational times. These instabilities are explained by the fact that dichotomic arc-flow branching may create a large number of nodes if decisions of this type are taken early in the tree (i.e., close to the root node). In this case, the probability of branching again on an arc is high, leading to an exponentiel growth of the tree and large computational times. The performance of CFAB also differs a lot from one instance to another. For instance 1, it finds solutions with gaps similar to those obtained by DCF, in approximately twice the time. For instance 2, the solutions found by CFAB have gaps close to those produced by RB, with computational times that are more than 150% larger on average. For instance 3, CFAB finds solutions with gaps more than 50% smaller than any other branching method, but in computational times that are more than 13 times larger.

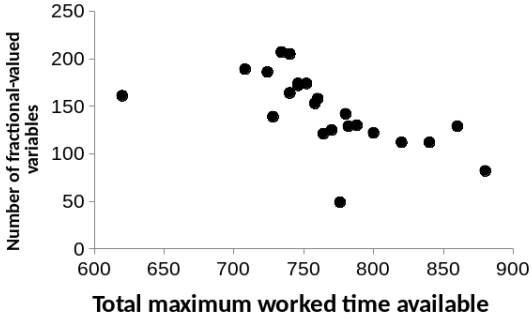
The RB heuristic yields better solutions than any other tested heuristic, with an average gap of 0.888% for instance 1 and 0.565% for instance 2. Furthermore, for highly constrained scenarios, RB produces gaps 35% smaller for instance 1 and 48% smaller for instance 2 with respect to those obtained by DCF. These improvements come however at the expense of a large increase in the average computational time, namely, 851% for instance 1, and 576% for instance 2. This can be considered acceptable since the average computational time is below 5 minutes, and only two scenarios yield a computational time exceeding 15 minutes.

Table 3: Results for instance 1

Scenario	DCF		SCF		CFAB		RB	
	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)
Unlimited	11.2	0.376	17.9	0.586	15.3	0.394	67.2	0.448
175/550/175	16.1	0.511	12.8	0.477	17.0	0.395	37.4	0.511
135/510/135	22.2	0.341	20.1	0.275	11.9	0.530	47.4	0.387
125/490/125	22.7	0.519	11.2	0.901	20.4	0.541	76.4	0.345
120/480/120	12.7	0.685	12.1	0.686	87.8	0.548	58.4	0.636
115/470/115	23.2	0.587	17.5	1.258	27.3	0.393	54.6	0.570
110/460/110	15.3	1.132	19.9	1.132	23.6	0.223	99.4	0.772
Average	17.6	0.593	15.9	0.759	29.0	0.432	63.0	0.524
105/450/105	21.7	1.422	33.9	1.759	66.7	1.415	72.1	0.432
100/440/100	23.1	0.992	27.8	1.207	34.6	1.556	330.8	0.925
95/430/95	24.5	0.891	16.5	0.893	13.4	1.174	426.0	1.193
90/420/90	15.8	0.853	31.0	0.853	13.7	0.932	292.0	0.850
89/410/89	15.4	0.737	17.7	0.994	16.4	0.835	142.7	0.938
88/400/88	17.3	0.758	14.2	0.859	14.0	0.782	82.5	0.767
87/390/87	16.9	0.868	13.8	0.875	15.5	1.130	58.0	0.726
86/380/86	19.2	0.938	10.9	0.669	17.2	0.812	140.8	0.696
100/340/107	17.1	0.526	16.9	1.408	12.7	0.953	31.4	0.514
85/370/85	14.6	0.647	15.3	0.502	18.3	0.700	114.9	0.672
98/335/104	16.7	1.363	20.8	0.698	22.2	1.751	133.7	1.333
84/360/84	17.8	1.578	12.8	0.665	20.2	0.785	76.4	0.619
94/330/101	23.1	2.013	36.3	1.013	507.1	1.876	313.3	1.537
83/350/83	20.1	0.707	23.5	1.209	13.6	0.541	100.1	0.549
92/325/98	28.5	1.346	45.3	0.580	24.0	1.331	249.2	1.150
90/320/95	23.4	2.370	18.6	3.043	15.7	1.390	134.8	1.016
82/340/82	17.5	0.728	24.4	0.690	12.7	0.598	85.7	0.656
Average	19.6	1.102	22.3	1.054	49.3	1.092	163.8	0.857
88/315/92	30.3	2.373	23.6	0.804	13.5	1.143	48.6	1.025
81/330/81	16.1	0.555	25.4	1.917	13.6	0.851	41.8	0.525
86/310/89	26.5	2.141	39.9	3.151	17.0	2.190	107.9	1.501
80/320/80	27.3	1.678	23.2	0.536	20.3	1.449	71.2	0.664
84/305/86	19.6	1.530	34.2	1.817	26.5	3.200	975.9	1.859
79/310/79	33.9	3.191	28.3	1.445	14.7	1.417	490.0	0.988
82/300/83	22.4	1.830	46.1	1.810	19.7	2.761	562.2	1.871
80/295/80	24.0	2.389	25.0	2.547	108.0	2.006	717.7	1.726
Average	25.0	1.961	30.7	1.753	29.2	1.877	376.9	1.270
Global average	20.5	1.205	23.0	1.164	39.8	1.144	195.0	0.888



(a) Number of branching nodes in the LP solution vs Total maximum worked time



(b) Number of fractional-valued variables in the LP solution vs Total maximum worked time

Figure 3: Impact of the total maximum worked time for instance 2

Table 4: Results for instance 2

Scenario	DCF		SCF		CFAB		RB	
	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)
Unlimited	31.5	0.370	29.2	0.326	81.6	0.228	29.8	0.177
200/600/80	29.0	0.312	46.9	0.597	108.2	0.114	58.4	0.255
193/590/77	30.1	0.357	29.4	0.284	189.2	0.209	56.6	0.414
186/580/74	39.8	0.627	28.0	0.346	1455.1	0.279	39.5	0.262
179/570/71	29.7	0.324	34.4	0.399	103.6	0.214	60.2	0.329
172/560/68	32.0	0.393	25.3	0.317	692.0	0.266	50.0	0.255
178/547/63	31.6	0.346	40.2	0.360	39.3	0.182	60.9	0.412
176/544/62	26.1	0.510	31.3	0.347	2528.6	0.244	111.0	0.633
165/550/65	27.9	0.534	35.2	0.573	42.8	0.252	49.5	0.301
174/541/61	30.8	0.419	36.2	0.491	1312.8	0.251	41.3	0.207
172/538/60	29.1	0.258	32.6	0.258	3612.1	0.177	109.7	0.497
Average	30.7	0.405	33.5	0.391	924.1	0.220	60.6	0.340
170/535/59	35.5	1.402	37.2	1.240	83.4	0.445	114.4	0.560
158/540/62	39.5	0.336	49.1	0.810	61.2	0.476	142.2	0.562
168/532/58	41.3	0.833	54.1	0.718	83.7	0.358	132.3	0.619
166/529/57	44.1	0.305	47.9	1.370	33.6	0.719	94.2	0.308
164/526/56	44.5	0.331	41.2	0.942	42.5	0.835	77.3	0.294
161/524/61	49.4	0.900	43.4	0.900	105.6	0.779	211.0	0.606
151/530/59	44.0	0.909	64.4	1.153	31.1	1.081	133.6	0.546
162/523/55	48.8	1.368	55.5	1.341	33.8	0.588	391.4	0.852
Average	43.4	0.798	49.1	1.059	59.4	0.660	162.1	0.543
160/520/54	50.0	1.469	54.5	1.384	48.4	0.762	432.3	0.646
158/517/53	71.1	2.590	84.3	1.505	400.5	1.412	921.5	1.113
148/520/56	58.2	1.723	62.4	1.430	170.8	1.085	2013.4	1.400
145/510/53	78.3	1.567	111.1	2.163	61.9	1.473	905.2	1.172
100/420/100	69.1	3.197	94.7	1.518	75.3	1.043	597.6	1.143
Average	65.3	2.109	81.4	1.600	151.4	1.155	974.0	1.095
Global average	42.1	0.891	48.7	0.866	474.9	0.561	284.7	0.565

Table 5: Results for instance 3

Scenario	DCF		SCF		CFAB		RB	
	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)
Unlimited	11.1	0.490	10.6	0.557	13.7	0.222	30.9	0.563
175/147/232	16.2	0.602	16.7	0.238	22.6	0.288	29.0	0.521
173/146/230	22.9	0.563	25.0	0.464	14.0	0.129	28.2	0.249
171/145/227	19.8	0.315	16.5	0.268	21.4	0.110	22.4	0.230
169/143/225	17.7	0.501	18.9	0.501	14.9	0.126	25.6	0.331
167/142/222	16.3	0.330	12.8	0.330	18.9	0.127	20.8	0.421
165/140/220	15.2	0.448	13.1	0.496	22.3	0.088	20.5	0.177
163/139/217	13.4	0.160	12.9	0.160	14.9	0.160	14.3	0.062
161/137/215	17.4	0.490	12.5	0.322	13.4	0.091	12.7	0.171
159/136/212	14.1	0.197	15.9	0.197	13.6	0.093	13.7	0.006
157/134/210	13.0	0.306	12.6	0.299	22.0	0.003	11.1	0.165
155/133/207	13.0	0.228	10.9	0.228	15.5	0.084	11.8	0.267
153/131/205	9.8	0.073	12.4	0.000	20.8	0.101	10.0	0.163
151/130/202	10.1	0.162	10.0	0.162	20.0	0.063	15.5	0.257
149/128/200	18.1	0.038	13.6	0.214	15.0	0.076	10.2	0.318
147/127/197	15.1	0.169	11.2	0.309	25.7	0.000	10.4	0.210
145/125/195	19.0	0.474	17.9	0.300	33.0	0.074	15.3	0.000
143/124/192	18.3	0.332	15.1	0.332	84.6	0.177	30.2	0.332
141/122/190	20.7	0.474	20.9	0.430	64.2	0.250	47.3	0.493
139/121/187	22.6	1.113	22.0	1.114	1326.2	0.424	114.2	1.388
137/119/185	21.6	0.970	21.7	0.970	6705.5	0.481	161.2	1.436
Global average	16.4	0.402	15.4	0.376	404.9	0.151	31.2	0.370

5.3.2 Large instances

The results obtained by the DCF, SCF and RB heuristics for the large instances are given in Tables 6-9. We make the following observations. The DCF method almost always finds solutions with gaps below 1%. However scenario 2390/1580/815 of instance 6 and scenario 1905/1620/230 of instance 4 exhibit gaps of 11.508% and 3.590%, respectively. These outliers illustrate a weakness of DCF : Sometimes fixing one or several columns can result in a dramatical increase of the objective value, and there is no way to review these "bad" decisions. In the remainder of this paper, average results concerning instances 4 and 6 are computed taking into account these bad results for DFC. Averages without them are also given in Tables 6 and 8.

The gaps obtained with the SCF heuristic are approximately 10% smaller for instance 5, and 30% smaller for instances 4 and 6, with respect to those obtained by DCF. The computational times are, however, slightly larger on average. For instance 7, the gaps obtained were similar (only 3.2% larger on average), with a 7.2% average increase of the computational times. Overall, SCF is preferable to DCF since the gain in solution quality outweighs the increase in computational times.

The RB method finds solutions with smaller gaps than any other tested method in a majority of scenarios. Furthermore, the average gaps are at least 25% smaller than those derived by DCF for instances 4 to 7, and at least 13% smaller than those obtained by SCF for instances 5 to 7. For instance 4, SCF finds solutions with gaps slightly smaller than those yielded by RB, but, on average, the computational times are approximately 11% larger. It is also worth noting that, for all instances and all scenarios, RB consistently computes solutions with gaps less than 0.9%. This is explained by the ability of RB to revise poor decisions when the objective value becomes larger than the estimated gap, and expresses the reliability of the method. The computational times of the RB heuristic are comparable to those of the DCF heuristic for instances 4 and 6, and are on average 46% and 39% larger for instances 5 and 7, respectively. These results contrast with the high relative difference in computational times reported for the small instances. We noticed that, when solving large instances, RB creates few sibling nodes, resulting in narrower trees. In this context, RB can be viewed as an extension of DCF with a safeguard mechanism that can correct retroactively bad branching decisions.

Table 6: Results for instance 4

Scenario	DCF		SCF				RB			
	Time (s)	Gap (%)	Time (s)	vs. DCF	Gap (%)	vs. DCF	Time (s)	vs. DCF	Gap (%)	vs. DCF
Unlimited	791	0.074	1103	39.5%	0.015	-79.7%	596	-24.6%	0.001	-98.6%
2150/1740/300	547	0.023	546	-0.1%	0.003	-87.0%	694	26.9%	0.035	52.2%
2100/1720/295	505	0.000	498	-1.4%	0.001	∞ %	824	63.2%	0.003	∞ %
2050/1700/285	748	0.180	804	7.5%	0.130	-27.8%	899	20.2%	0.149	-17.2%
2025/1695/275	572	0.165	582	1.7%	0.165	0.0%	570	-0.4%	0.108	-34.5%
2000/1685/270	690	0.145	743	7.7%	0.064	-55.9%	723	4.7%	0.026	-82.1%
1980/1665/265	1280	0.660	1667	30.2%	0.524	-20.6%	955	-25.4%	0.342	-48.2%
1965/1645/260	1144	0.424	1415	23.7%	0.319	-24.8%	1290	12.7%	0.312	-26.4%
1940/1630/255	1400	0.718	1350	-3.6%	0.346	-51.8%	913	-34.8%	0.220	-69.4%
1920/1615/245	1012	0.111	1014	0.1%	0.111	0.0%	1133	11.9%	0.487	338.7%
1890/1695/220	1081	0.235	1318	21.9%	0.189	-19.6%	1356	25.5%	0.341	45.1%
1905/1620/230	956	3.590	956	-0.0%	3.590	0.0%	1097	14.8%	0.023	-99.4%
Average	894	0.527	999	11.8%	0.455	-13.7%	921	3.0%	0.171	-67.6%
Average without 1905/1620/230	888	0.249	1003	13.0%	0.170	-31.7%	905	1.9%	0.184	-26.0%

Table 7: Results for instance 5

Scenario	DCF		SCF				RB			
	Time (s)	Gap (%)	Time (s)	vs. DCF	Gap (%)	vs. DCF	Time (s)	vs. DCF	Gap (%)	vs. DCF
Unlimited	580	0.188	632	9.1%	0.241	28.2%	532	-8.3%	0.201	6.9%
1790/1000/540	764	0.382	896	17.2%	0.324	-15.2%	679	-11.2%	0.253	-33.8%
1780/990/535	769	0.431	975	26.8%	0.361	-16.2%	570	-26.0%	0.242	-43.9%
1770/980/530	913	0.550	1146	25.5%	0.357	-35.1%	1404	53.8%	0.595	8.2%
1760/970/525	687	0.948	747	8.6%	0.341	-64.0%	809	17.7%	0.388	-59.1%
1750/960/520	889	0.587	822	-7.5%	0.469	-20.1%	1355	52.5%	0.527	-10.2%
1740/950/515	948	1.291	934	-1.5%	1.029	-20.3%	2258	138.2%	0.844	-34.6%
1730/940/510	973	0.727	1067	9.7%	0.814	12.0%	963	-0.9%	0.609	-16.2%
1720/930/505	1154	0.575	2007	73.9%	1.237	115.1%	3573	209.6%	0.702	22.1%
1710/920/500	910	0.759	930	2.2%	0.546	-28.1%	1169	28.5%	0.349	-54.0%
1700/910/495	1097	0.446	1087	-0.9%	0.523	17.3%	898	-18.1%	0.584	30.9%
1690/900/490	1227	0.801	1541	25.6%	0.514	-35.8%	1768	44.2%	0.417	-47.9%
Average	909	0.640	1065	17.2%	0.563	-12.1%	1331	46.5%	0.476	-25.7%

Table 8: Results for instance 6

Scenario	DCF		SCF				RB			
	Time (s)	Gap (%)	Time (s)	vs. DCF	Gap (%)	vs. DCF	Time (s)	vs. DCF	Gap (%)	vs. DCF
Unlimited	620	0.032	642	4%	0.030	-6%	634	2%	0.120	275%
2515/1660/855	1530	0.270	1482	-3%	0.108	-60%	1281	-16%	0.341	26%
2500/1650/845	1914	1.084	1645	-14%	0.445	-59%	1489	-22%	0.368	-66%
2480/1640/840	1444	0.403	1647	14%	0.806	100%	1393	-4%	0.425	6%
2460/1625/835	1997	0.223	2138	7%	0.414	86%	1338	-33%	0.140	-37%
2445/1610/830	1397	0.246	1544	11%	0.187	-24%	1273	-9%	0.140	-43%
2425/1600/825	1901	0.404	1817	-4%	0.267	-34%	1569	-18%	0.132	-67%
2410/1590/820	1270	0.068	1236	-3%	0.068	0.0%	1418	12%	0.240	253%
2400/1585/815	2054	0.365	2437	19%	0.258	-29%	1761	-14%	0.209	-43%
2390/1580/815	2135	11.508	2293	7%	0.392	-97%	1531	-28%	0.154	-99%
2385/1570/810	1706	0.123	1639	-4%	0.170	38%	2569	51%	0.280	128%
2375/1565/810	2065	1.315	1941	-6%	0.405	-69%	2117	3%	0.324	-75%
Average	1669	1.337	1705	2%	0.296	-78%	1531	-8%	0.239	-82%
Average without 2390/1580/815	1627	0.412	1652	2%	0.287	-30%	1531	-6%	0.247	-40%

Table 9: Results for instance 7

Scenario	DCF		SCF				RB			
	Time (s)	Gap (%)	Time (s)	vs. DCF	Gap (%)	vs. DCF	Time (s)	vs. DCF	Gap (%)	vs. DCF
Unlimited	798	0.572	873	9.4%	0.290	-49.3%	913	14.4%	0.334	-41.6%
610/1220/360	591	0.276	546	-7.7%	0.276	0.0%	738	24.8%	0.245	-11.2%
605/1210/360	671	0.262	633	-5.6%	0.247	-5.7%	710	5.8%	0.355	35.5%
600/1200/355	1103	0.751	1537	39.3%	0.542	-27.8%	677	-38.6%	0.479	-36.2%
595/1190/350	1065	0.644	1191	11.8%	0.884	37.3%	1756	64.8%	0.506	-21.4%
590/1180/350	1265	1.656	1817	43.6%	0.932	-43.7%	3201	153.1%	0.583	-64.8%
585/1170/350	1176	0.624	1240	5.5%	0.638	2.2%	1342	14.1%	0.569	-8.8%
580/1160/345	1354	0.410	1337	-1.2%	0.371	-9.5%	2073	53.1%	0.467	13.9%
575/1150/340	1318	0.869	1271	-3.6%	0.835	-3.9%	1352	2.6%	0.353	-59.4%
570/1140/340	1517	0.459	1586	4.5%	0.351	-23.5%	1712	12.8%	0.541	17.9%
565/1130/340	2057	0.530	1931	-6.1%	1.168	120.4%	2916	41.7%	0.532	0.4%
560/1120/335	1807	0.666	1831	1.3%	1.433	115.2%	3139	73.7%	0.549	-17.6%
Average	1227	0.643	1316	7.3%	0.664	3.2%	1711	39.4%	0.459	-28.6%

5.4 Sensitivity analysis

Given that RB is a new branching scheme that we introduce in this paper, we present a sensitivity analysis on the 6 parameters controlling this branching heuristic. For each parameter, we ran the RB heuristic on all tested instances and all scenarios using one or two other values surrounding the value given in Table 2 and used to provide the results in the previous section. One parameter value is changed at a time. Table 10 reports the average computational times and gaps obtained from these experiments. The row identified by *Best* indicates the selected parameter setting. The parameters listed in the first column specify for each test which parameter value varied. For the small instances, values of τ^C higher than 0.9 were not tested since the threshold would be so high almost no columns would be ever fixed. Similarly, the case $\nu^R = 0$ was not tested for the large instances because it would make the RB heuristic equivalent to the DCF method.

Table 10: Sensitivity analysis on the parameters of the RB heuristic

	Small instances			Large instances		
	$(\tau^C, \mu^C, \xi^C, E^R, \Delta^R, \nu^R)$	Time (s)	Gap (%)	$(\tau^C, \mu^C, \xi^C, E^R, \Delta^R, \nu^R)$	Time (s)	Gap (%)
Best	(0.9, 2, 0.4, 0.3, 0.3, 2)	170.3	0.607	(0.7, ∞, 1.5, 0.2, 0.1, 1)	1373.4	0.336
τ^C	(0.8, 2, 0.4, 0.3, 0.3, 2)	139.9	0.610	(0.6, ∞, 1.5, 0.2, 0.1, 1)	986.6	0.541
				(0.8, ∞, 1.5, 0.2, 0.1, 1)	6094.8	0.400
μ^C	(0.9, 1, 0.4, 0.3, 0.3, 2)	448.0	0.601	(0.7, 10, 1.5, 0.2, 0.1, 1)	1795.3	0.356
	(0.9, 3, 0.4, 0.3, 0.3, 2)	176.5	0.608	(0.7, 5, 1.5, 0.2, 0.1, 1)	2836.7	0.362
ξ^C	(0.9, 2, 0.3, 0.3, 0.3, 2)	180.4	0.607	(0.7, ∞, 1, 0.2, 0.1, 1)	1962.8	0.350
	(0.9, 2, 0.5, 0.3, 0.3, 2)	179.8	0.607	(0.7, ∞, 2, 0.2, 0.1, 1)	1252.4	0.361
E^R	(0.9, 2, 0.4, 0.2, 0.3, 2)	196.3	0.588	(0.7, ∞, 1.5, 0.1, 0.1, 1)	1548.7	0.329
	(0.9, 2, 0.4, 0.4, 0.3, 2)	151.9	0.644	(0.7, ∞, 1.5, 0.3, 0.1, 1)	1251.2	0.351
Δ^R	(0.9, 2, 0.4, 0.3, 0.2, 2)	236.6	0.590	(0.7, ∞, 1.5, 0.2, 0.05, 1)	1607.5	0.308
	(0.9, 2, 0.4, 0.3, 0.4, 2)	137.5	0.623	(0.7, ∞, 1.5, 0.2, 0.2, 1)	1259.1	0.356
ν^R	(0.9, 2, 0.4, 0.3, 0.3, 1)	79.4	0.714	(0.7, ∞, 1.5, 0.2, 0.1, 2)	2519.1	0.329
	(0.9, 2, 0.4, 0.3, 0.3, 3)	355.5	0.585			

From these results, we first remark that the selected parameter setting is Pareto-optimal with regards to the average computational time and gap for both groups of instances. The results also highlight that the values of the parameters E^R and Δ^R are crucial to obtain good results. Indeed, the best way to obtain good-quality solutions is to underestimate the expected gap and update its value by small amounts when needed.

For the small instances, the maximum number of columns fixed at each node (μ^C) has the most important impact on the computational times, as fixing columns one at a time causes the average computational time to almost triple. A larger value of ν^R results in smaller gaps, but larger computational times. The solutions found have significantly smaller gaps when $\nu^R = 2$ compared to $\nu^R = 1$. Using $\nu^R = 3$ yields a much smaller gain in solution quality, but substantially increases the average computational time. This indicates that, for the small instances, large increases of the objective value are sometimes caused by a combination of two risky columns, but rarely of three such columns or more.

For the large instances, the parameter τ^C has the largest impact on the computational times. If τ^C is too large, less columns are fixed at each node, and columns are more frequently added to the risky column set, potentially creating a large search tree. Parameters μ^C and ξ^C also impact the computational times, as fixing too few columns at each node results in a deeper tree. Finally, we observe that, for these instances, $\nu^R = 1$ offers the best compromise between solution quality and the computational times.

6 Conclusions

In this paper, we proposed a new branching scheme, called retrospective branching (RB), that can be used in a column generation framework to solve the CPPBC. We compared the performance of this heuristic with three exiting branching heuristics used in the airline industry. Our computational results show that the RB heuristic performs better than the other tested heuristics, with significantly smaller gaps between the value of the computed integer solution and the value of the computed linear relaxation solution, or shorter computational times. The RB method performs especially well when the maximum worked time allocated to the bases is very tight. We also observed that RB is more reliable at finding good-quality solutions in reasonable times than the other branching heuristics.

An extension of this research would be to consider the monthly version of the CPPBC. This would pose an additional challenge because the larger size of the monthly instances would result in much larger computational times. It would also be interesting to study the impact of the pairings produced by solving the CPPBC on the schedules generated in the CAP phase. Finally, the RB heuristic could be applied to a different problem solved by a branch-and-price heuristic.

References

[1] Saddoune M, Desaulniers G, Elhallaoui I, Soumis F. Integrated Airline Crew Pairing and Crew Assignment by Dynamic Constraint Aggregation. Transportation Science 2012;46(1):39–55. doi:10.1287/trsc.1110.0379.

[2] Hu J, Johnson EL. Computational results with a primaldual subproblem simplex method. Operations Research Letters 1999;25(4):149–57. doi:http://dx.doi.org/10.1016/S0167-6377(99)00048-6.

[3] Klabjan D, Johnson EL, Nemhauser GL, Gelman E, Ramaswamy S. Solving Large Airline Crew Scheduling Problems: Random Pairing Generation and Strong Branching. Computational Optimization and Applications 2001;20(1):73–91. doi:10.1023/A:1011223523191.

[4] Soykan B, Erol S. A Branch-and-Price Algorithm for the Robust Airline Crew Pairing Problem. 2014. doi:10.17134/sbd.88400.

[5] Irnich S, Desaulniers G. Shortest path problems with resource constraints. Column generation 2005;33–65.

[6] Lozano L, Medaglia AL. On an exact method for the constrained shortest path problem. Computers & Operations Research 2013;40(1):378–84. doi:10.1016/j.cor.2012.07.008.

[7] Mercier A, Soumis F. An integrated aircraft routing, crew scheduling and flight retiming model. Computers & Operations Research 2007;34(8):2251–65.

[8] Cacchiani V, Salazar-González JJ. Optimal Solutions to a Real-World Integrated Airline Scheduling Problem. Transportation Science 2015. doi:10.1287/trsc.2015.0655.

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

[10] Gopalakrishnan B, Johnson E. Airline Crew Scheduling: State-of-the-Art. Annals of Operations Research 2005;140(1):305–37. doi:10.1007/s10479-005-3975-3.

[11] Anbil R, Forrest JJ, Pulleyblank WR. Column generation and the airline crew pairing problem. Documenta Mathematica 1998;3(1):677.

[12] Desaulniers G, Desrosiers J, Dumas Y, Marc S, Rioux B, Solomon M, et al. Crew pairing at Air France. European Journal of Operational Research 1997;97(2):245–59. doi:10.1016/S0377-2217(96)00195-6.

[13] Zeren B, Özkol I. A novel column generation strategy for large scale airline crew pairing problems. Expert Systems with Applications 2016;55:133–44. doi:10.1016/j.eswa.2016.01.045.

[14] Joncour C, Michel S, Sadykov R, Sverdlov D, Vanderbeck F. Column Generation based Primal Heuristics. Electronic Notes in Discrete Mathematics 2010;36:695–702. doi:10.1016/j.endm.2010.05.088.

[15] Muter I, Birbil SI, Bülbül K, ahin G, Yenigün H, Ta D, et al. Solving a robust airline crew pairing problem with column generation. Computers & Operations Research 2013;40(3):815–30.

[16] Aydemir-Karadag A, Dengiz B, Bolat A. Crew pairing optimization based on hybrid approaches. Computers & Industrial Engineering 2013;65(1):87–96. doi:10.1016/j.cie.2011.12.005.

[17] Barnhart C, Johnson E, Nemhauser G, Savelsbergh M, Vance PH. Branch-and-price: Column generation for solving huge integer programs. Operations Research 1998;46(3):316–29.

[18] Desaulniers G, Desrosiers J, Solomon M. Column generation. Springer, New York, NY; 2005.

[19] Villeneuve D, Desaulniers G. The shortest path problem with forbidden paths. European Journal of Operational Research 2005;165(1):97–107. doi:10.1016/j.ejor.2004.01.032.

[20] Saddoune M, Desaulniers G, Soumis F. Aircrew pairings with possible repetitions of the same flight number. Computers and Operations Research 2013;40(3):805–14. doi:10.1016/j.cor.2010.11.003.

[21] Kasirzadeh A, Saddoune M, Soumis F. Airline crew scheduling: models, algorithms, and data sets. EURO Journal on Transportation and Logistics 2015;1:1–27. doi:10.1007/s13676-015-0080-x.