Internship subject : Crew rostering problem

Current report

Start date : 01/04/2024

End date : 15/09/2024

Location : RPTU Rheinland-Pfälzische Technische Universität (Kaiserslautern, Germany)

Intern : Mohamed HAMDI ABDALA

Supervisor : Marvin CASPAR

Academic tutor : Mahdi MOEINI

Contents

August 26, 2024

Abstract

With the continuous growth of the airline industry, companies are expanding their fleets to increase market share. This expansion leads to a rise in the complexity and scale of the crew rostering problem, a major planning challenge in the industry. Consequently, significant resources are allocated to address this problem, often involving the use of costly external software solutions. This report provides an overview of the crew rostering problem, including a literature review, a detailed problem formulation, and proposed resolution methods. Additionally, potential future research directions are discussed to develop more efficient and effective solutions for crew rostering.

1 Introduction

The crew rostering problem is a critical and complex challenge faced by the airline industry. It involves scheduling crew members (pilots, co-pilots, flight attendants,....) to ensure all flights are adequately staffed while adhering to various regulations and constraints. This problem is significant not only due to its impact on operational efficiency and costs but also because it directly affects the safety and satisfaction of both crew members and passengers.

At its core, the crew rostering problem requires assigning crew members to a set of predefined pairings over a specific planning horizon, typically a month. A pairing is a sequence of flights and rest periods that starts and ends at a crew base and is generated in a prior phase known as the crew pairing problem. The goal of the crew rostering problem is to ensure that all generated pairings are covered, while complying with labor laws, contractual agreements, and considering crew preferences. The sequence of pairings separated by time off that covers the given time horizon is called a roster. The primary goal of crew rostering is to assign each roster to a crew member, ensuring a balanced distribution of work among crew members while maximizing their satisfaction and adherence to regulatory standards.

Given the multifaceted nature of the crew rostering problem, a thorough understanding of existing methodologies and their limitations is essential. In this internship, we decided to begin with a literature review to explore various approaches that have been developed and applied in this domain. Following this, we presented a precise formulation of our specific crew rostering problem, detailing the key variables, constraints, and objectives involved. Finally, we discussed the proposed methods for resolving the problem, emphasizing both traditional and innovative techniques aimed at improving the efficiency and effectiveness of crew scheduling.

2 Review on the Crew Rostering Problem

The crew scheduling challenge is typically divided into two sequential phases: crew pairing and crew rostering. In the crew pairing problem (CPP), the goal is to select feasible pairings from a large set while minimizing costs, ensuring that all flights are covered exactly once, and adhering to pairing regulations and contractual rules. Subsequently, in the crew rostering problem (CRP), crews are assigned to each selected pairing, with specific requirements for crew positions (such as pilot, co-pilot, first officer, second officer, etc.). The objective here is to cover all pairings while ensuring compliance with airline regulations on crew composition. For the purpose of this internship, we assume that pairings have already been generated during the crew pairing phase, focusing primarily on the crew rostering phase.

Caprara et al. [4] propose a modeling and solution approach for the crew rostering problem, representing duties as nodes and legal transitions as edges, with the aim of minimizing total crew numbers while ensuring the validity of rosters. Kohl and Karisch [9] investigate the assignment of crew pairings to either anonymous bidlines or personalized rosters, aiming to optimize cost efficiency and crew satisfaction. Kasirzadeh et al. [8] consider airline regulations and crew preferences, balancing the cost of personalized rosters with penalties for unmet preferences and constructed monthly schedules for pilots via a sequential approach based on branch and price.

Deveci and Demirel [5] provide a comprehensive survey of airline crew scheduling literature, covering key problems, methodologies, and advancements, from set covering and set partitioning formulations to more complex models such as Mixed Integer Programming (MIP), non-linear, and stochastic approaches. Walid el Moudani et al. [6] propose a bi-criterion approach for personalized crew rostering, taking into account pilots' capabilities, maximum flight times, and the number of pairings, with the objective of maximizing satisfaction while controlling costs.

Parmentier and Meunier [10] address the crew pairing problem, introducing non-linear constraints related to the proportion of long and short duties and pairings, aiming to minimize pairing costs using a column generation approach. Xu et al. [12] integrate crew pairing and rostering to maximize the number of satisfied vacation requests and preferred flights while minimizing total pairing costs. Quesnel et al. [11] incorporate deep-learning-based partial pricing in a branch-and-price algorithm, balancing overall crew satisfaction and penalties for violating constraints related to language proficiency and crew member counts per pairing.

The subsequent sections elaborate on the problem formulation, the methodologies employed to generate feasible rosters, and the assignment strategies aimed at achieving optimal solutions that satisfy both crew preferences and regulatory constraints.

3 Problem Formulation

In most European airlines, crew member satisfaction and fairness are crucial factors during the crew rostering phase. Prior to assigning schedules, airlines often solicit crew preferences regarding flights and days off. Due to insufficient preference data, a general criterion must be established to gauge the attractiveness of a roster.

To solve the crew rostering problem, the 'generate-and-optimize' principle is widely applied. This approach involves two primary phases: generating feasible rosters that adhere to airline regulations, followed by the assignment phase where crew members are selected for each roster to maximize satisfaction, fairness, and regulatory compliance.

Let R denote the set of all legal rosters. For example, if $R = \{ R_1 = [P^1, P^2, P^3], R_2 = [P^5, P^6] \}$, then pairings P^1, P^2, P^3 and P^5, P^6 must adhere to compatibility constraints within the same base without conflict. Additionally, R_1 and R_2 must conform to specific regulations outlined for this internship.

Rule	**Definition**	**Reference**
Global flown time	Computed from flights in pairings, monthly capped ($\leq F_{max}$) (hours)	[6], [11], [9], [8]
Pairings number	Each roster's number of pairings is bounded ($\leq P_{max}$)	[6]
Working days	Maximum number of working days in a roster bounded ($\leq W_{max}$)	[8]
Days off	Minimum number of rest days in a roster bounded ($\geq Off_{min}$)	[11], [9], [8]
Rest between pairings	Minimum rest time between successive pairings ($\geq \mathrm{R}_{min}$)	[11], [9]
Consecutive working days	Maximum number of consecutive working days in a roster bounded ($\leq CW_{max}$)	[11]

The constraints for generating a roster can be formulated as follows:

$$\sum_{p \in P} f_p \cdot y_p \leq F_{max} \tag{1}$$

$$\sum_{p \in P} y_p \leq P_{max} \tag{2}$$

$$\sum_{d \in D} z_d \geq Off_{min} \tag{3}$$

$$y_{p1} + y_{p2} \leq 1 \qquad \text{If rest time between } p1 \text{ and } p2 \text{ is } < R_{min} \tag{4}$$

$$y_{p1} + y_{p2} \leq 1 \qquad \text{If } p1 \text{ and } p2 \text{ don't have the same base} \tag{5}$$

$$\sum_{d \in CD} z_d \geq 1 \qquad \forall \text{ set of consecutive days } CD \text{ s.t } |\text{CD}| = \text{CW} \tag{6}$$

$$y_p + z_d \leq 1 \qquad \text{If } d \text{ is a working day for } p \tag{7}$$

$$y_p \in \{0, 1\} \qquad \forall p \in P \tag{8}$$

$$z_d \in \{0, 1\} \qquad \forall d \in D$$

Thus, a roster is defined by its included pairings (y_p indicates if p is included in the roster) and optional off days (z_d indicates if d is a roster off day). The model could omit the z_d variable to simplify, albeit potentially increasing complexity. In constraint gc1, f^p denotes the flown time in pairing p.

Using these constraints, legal rosters are generated (e.g., via column generation) and assigned to crew members. Before introducing the crew assignment model, key parameters include:

- **Parameters**:

L : Set of crew members (or pilots)

P : Set of pairings generated after the crew pairing phase

R : Set of all generated rosters/schedules

U : Set of crew position names (pilot, co-pilot, first officer, ...)

L_u $(\forall u \in U)$: Set of crew members in position u

U_p $(\forall p \in P)$: Required crew positions for pairing p

$a_{rp} = 1$: If pairing p is included in roster r, otherwise 0

$\delta_{lr} = 1$: If l and r are NOT in the same base, otherwise 0

C_l^r : "Cost" of roster r in terms of dissatisfaction

Here is a possible problem formulation:

$$\text{Minimize} \quad \sum_{l \in L} \sum_{r \in R} C_l^r \cdot x_l^r \tag{9}$$

$$\text{subject to} \quad \sum_{r \in R} x_{lr} \leq 1 \qquad \forall l \in L \tag{10}$$

$$\sum_{l \in L_u} \sum_{r \in R} a_{rp} \cdot x_{lr} = n_{pu} \qquad \forall p \in P,\ u \in U_p \tag{11}$$

$$\sum_{r \in R} \delta_{lr} \cdot x_{lr} = 0 \qquad \forall l \in L \tag{12}$$

$$x_{lr} \in \{0, 1\} \qquad \forall l \in L, r \in R$$

As described in the model, objective function eq:minimization minimizes total assignment costs, which typically reflect roster characteristics such as pairings and days off. Constraint mpc1 ensures each crew member is assigned to at most one legal roster, constraint mpc2 ensures that each pairing requirements are covered (n_{pu} is the required number of members which position is u in the crew of pairing p), and constraint mpc3 ensures crew members are not assigned to rosters from different bases.

4 First Approach

4.1 Generate-and-Optimize

In this approach, we have two phases:

- In the first phase, we generate all feasible rosters that comply with airline regulations. Here, we define a legal roster using an Integer Programming model, where the rules are characterized by constraints. There is no objective function in this phase, as our goal is to generate all possible rosters. The program is formulated in Answer Set Programming (ASP) with the Clingo solver.

We recall that the linear formulation of the roster generation problem is as follows:

$$\sum_{p \in P} f_p \cdot y_p \leq F_{max} \tag{13}$$

$$\sum_{p \in P} y_p \leq P_{max} \tag{14}$$

$$\sum_{d \in D} z_d \geq Off_{min} \tag{15}$$

$$y_{p1} + y_{p2} \leq 1 \qquad \text{If rest time between } p1 \text{ and } p2 \text{ is } < R_{min} \tag{16}$$

$$y_{p1} + y_{p2} \leq 1 \qquad \text{If } p1 \text{ and } p2 \text{ don't have the same base} \tag{17}$$

$$\sum_{d \in CD} z_d \geq 1 \qquad \forall \text{ set of consecutive days } CD \text{ s.t } |\text{CD}| = \text{CW} \tag{18}$$

$$y_p + z_d \leq 1 \qquad \text{If } d \text{ is a working day for } p \tag{19}$$

$$y_p \in \{0, 1\} \qquad \forall p \in P \tag{20}$$

$$z_d \in \{0, 1\} \qquad \forall d \in D$$

Then we can formulate it in ASP as following :

Selected pairings variable :
{select_pairing(P)} :- pairing(P).

Days off variable :
{off(D)} :- day(D).

Same pairing_base constraint :
:- select_pairing(P1), select_pairing(P2), pairing_base(P1,B1), pairing_base(P2,B2), B1 != B2.

Max flown time constraint :
:- sum {H,P : select_pairing(P), f(P,H) } > fmax.

Max number of selected pairings constraint :
:- {select_pairing(P) : pairing(P)} > pmax.

:- {select_pairing(P) : pairing(P)} = 0.

Minimum number of days off constraint :
:- {off(D) : day(D)} < offmin.

Minimum rest time between pairings constraint :
:- select_pairing(P1), select_pairing(P2), s(P1,ST1), e(P1,ET1), s(P2,ST2), ST2 > ST1, ST2 - ET1 < rmin.

Incompatibility between days off and selected pairings constraint :
:- select_pairing(P), off(D), w(P,D).

Off day definition constraint :
off(D) :- {select_pairing(P): w(P,D)} = 0, day(D).

Max number of consecutive working days constraint :
:- {off(D2) : consset(D1,D2)} < 1, day(D1), m - D1 >= cwmax.

- **Variables**:

select_pairing(P): Indicates if the pairing **P** is in the generated roster.
off(D): Indicates if the **D** is an off day in the generated roster.

- **Constants**:

m: Number of days in the rostering period.
fmax: Max flown time (in hours) of a legal roster.
pmax: Max number of pairings in a legal roster.
offmin: Min number of rest/off days in a legal roster.
rmin: Min rest time (in minutes) between two pairings in a legal roster.
cwmax: Max number of consecutive working days in a legal roster.

- **Atoms**:

pairing: The pairings atoms
day: The days in the planning rostering period atoms
pairing_base(P,B): Indicates if **B** is the base of pairing **P**
f(P,H): Indicates if **H** is the flown time (in hours) of pairing **P**
s(P,H): Indicates if **H** is the starting time (in minutes) of pairing **P**
e(P,H): Indicates if **H** is the end time (in minutes) of pairing **P**
s(P,H): Indicates if **H** is the flown time (in hours) of pairing **P**
w(P,D): Indicates if **D** is a working in pairing **P**
connsset(D1,D2): Indicates if there are at most **cwmax** days between **D1** and **D2** (**D2** > **D1**), as well as between **D1** and the last day of the rostering period (**m**)

We use the Clingo API in Python to solve the generation problem, then with the generated rosters we solve the assignement problem :

$$\begin{aligned}
\text{Minimize} \quad & \sum_{l \in L} \sum_{r \in R} C_l^r \cdot x_l^r & & (21)\\
\text{subject to} \quad & \sum_{r \in R} x_{lr} \leq 1 & \forall l \in L & (22)\\
& \sum_{l \in L_u} \sum_{r \in R} a_{rp} \cdot x_{lr} = n_{pu} & \forall p \in P,\ u \in U_p & (23)\\
& \sum_{r \in R} \delta_{lr} \cdot x_{lr} = 0 & \forall l \in L & (24)\\
& x_{lr} \in \{0, 1\} & \forall l \in L, r \in R &
\end{aligned}$$

To maximize satisfaction, we need to define certain criteria for the cost formula C_l^r, primarily based on the preferences of the crew members. In the inputs, each crew member l has his favorite pairings FP_l and favorite rest days FD_l. We also have a weight W_l that characterizes the priority between crew members (for example, cockpit satisfaction is usually more important than cabin crew satisfaction). Then we define the following criterion:

$$C_l^r = \sum_{\substack{p \in r \\ p \notin FP_l}} \left(W_l \left\lfloor \frac{f_p}{4} \right\rfloor \right) + \sum_{\substack{d \in FD_l \\ d \notin r}} W_l$$

Essentially, if we assign a roster r to a crew member l, we penalize unwanted pairings and rest days in r. In the first term, the weights $W_l \left\lfloor \frac{f_p}{4} \right\rfloor$ are used because the minimum flown time of a pairing is 8 hours, and we attempt to 'normalize' this when compared to the satisfaction from rest days (the second term). Therefore, the objective is to minimize overall dissatisfaction while simultaneously maximizing fairness among crew members with the same positions, with the weights

4.2 Column Generation

With a large number of pairings and an extensive planning or rostering period, we can generate a vast number of rosters. However, many of these rosters may not be used, resulting in an unnecessarily large number of variables in our problem. To restrict the size of the problem, it is beneficial to consider a column generation approach.

To define our restricted master problem (RMP), we start with the set of all crew members L and assume that R' is the current set of rosters. Firstly, we formulate the associated dual problem:

$$\text{Max} \quad \sum_{l \in L} -v_l + \sum_{p \in P} \sum_{u \in U_p} n_{pu} w_{pu} \tag{25}$$

$$\text{subject to} \quad -v_l + \sum_{p \in P} \sum_{u \in U_p} a_{rp} w_{pu} + \delta_{lr} b_l \leq C_l^r \qquad \forall l \in L_u,\ r \in R \tag{26}$$

$$w_{pu} \in \mathbb{R} \qquad \forall p \in P,\ u \in U_p \tag{27}$$

$$v_l \geq 0 \qquad \forall l \in L \tag{28}$$

$$b_l \in \mathbb{R} \qquad \forall l \in L \tag{29}$$

Then we write the pricing problem in ASP as :

Selected pairings variable :
{select_pairing(P)} :- pairing(P).

Days off variable :
{off(D)} :- day(D).

Members selected constraint index:
{select_member(L)} :- member(L).

If the member don't have the same base as the generated pairing:
{delta(L)} :- member(L).

Select a new roster :
:- roster(R), {not select_pairing(P) : in-roster(R,P)} = 0, {select_pairing(P) : not in-roster(R,P)} = 0.

Select exactly one "1" (or member) index :
{select_member(L) : member(L)} = 1 :- .

Define delta in asp :
:- delta(L), member_base(L,B), pairing_base(P,B), select_pairing(P).
:- not delta(L), member_base(L,B1), pairing_base(P,B2), select_pairing(P), B1 != B2.

Same pairing_base constraint :
:- select_pairing(P1), select_pairing(P2), pairing_base(P1,B1), pairing_base(P2,B2), B1 != B2.

Max flown time constraint :
:- sum {H,P : select_pairing(P), f(P,H) } > fmax.

Max number of selected pairings constraint :
:- {select_pairing(P) : pairing(P)} > pmax. :- {select_pairing(P) : pairing(P)} = 0.

Minimum number of days off constraint :
:- {off(D) : day(D)} < offmin.

Minimum rest time between pairings constraint :
:- select_pairing(P1), select_pairing(P2), s(P1,ST1), e(P1,ET1), s(P2,ST2), ST2 > ST1, ST2 - ET1 < rmin.

Incompatibility between days off and selected pairings constraint :
:- select_pairing(P), off(D), w(P,D).

Off day definition constraint :
off(D) :- {select_pairing(P): w(P,D)} = 0, day(D).

Max number of consecutive working days constraint :
:- {off(D2) : consset(D1,D2)} < 1, day(D1), m - D1 >= cwmax.

minimize {
W*(H/4) : select_pairing(P), select_member(L), not favorite_pairings(L,P), weight(L,W), f(P,H);
W, D : not off(D), select_member(L), favorite_rest_days(L,D), weight(L,W);
V : select_member(L), dual_v(L,V);
-W : select_pairing(P), pairing_positions(P,U), select_member(L), member_position(L,U), dual_w(P,U,W);
-B: select_member(L), delta(L), dual_b(L,B).}

- **New variables**:

select_member(L): As the dual constraints are indexed by **r,l**, we need to select an index **l** (a crew member).
delta(L): Indicates if the **D** is an off day in the generated roster.

- **New atoms**:

member: The crew members atoms
roster: The current generated rosters atoms
in-roster(R,P): Indicates if pairing **P** is included in the current roster **R**
pairing_positions(P,U): Indicates if the position **U** (U can be for example 'pilot', 'copilot', 'purser', 'steward' and 'hostess' ...) is required for the pairing **P**. Obviously, some positions will always be required, this atom is used in case some a pairing require some of the occasional ones like Flight Engineer, Airborne Maintenance Technician, ...
member_position(L,U): Indicates if **U** is the actual position of the crew member **L**
favorite_pairings(L,P): Indicates if **P** is one of the favorite pairings of the crew member **L**
favorite_rest_days(L,D): Indicates if **D** is one of the favorite days of the crew member **L** (to be an off day)
dual_v(L,V): Indicates if **V** is the current dual solution associated to the first constraint of the primal problem
dual_w(L,W): Indicates if **B** is the current dual solution associated to the second constraint of the primal problem
dual_b(L,B: Indicates if **B** is the current dual solution associated to the third constraint of the primal problem

With these steps we can now write our Column generation algorithm:

Algorithm 1 SolveRMP

0: **Input:** Current set of columns
0: **Output:** Optimal solution and dual variables
0: Solve the linear programming problem defined by the current columns
0: **return** optimal solution and dual variables =0

Algorithm 2 SolvePricingProblem

0: **Input:** Dual variables from the RMP
0: **Output:** Column with the most negative reduced cost
0: Formulate the pricing problem using the dual variables
0: Solve the pricing problem to find the column with the most negative reduced cost
0: **return** the new column =0

Algorithm 3 Column Generation

0: Initialize RMP with initial feasible columns
0: **while** not Converged **do**
0: Solve the RMP to obtain the dual variables
0: Solve the pricing problem using the dual variables
0: **if** reduced cost < 0 **then**
0: Add new column to the RMP
0: **else**
0: Converged = true
0: **end if**
0: **end while**
0: **return** optimal solution from the RMP =0

5 Second approach: Sub-problems

If we look back at the master problem, we can observe that it is possible to decompose it into several subproblems where all crew members and pairings belong to the same base. In this part, we will remove the third constraint in the master problem ($\sum_{r\in R} \delta_{lr} \cdot x_{lr} = 0 \quad \forall l \in L$) and solve the subproblem where L and P are subsets of the crew members and pairing sets that correspond to the same base.

5.1 Generate-and-Optimize

From the linear formulation of the roster generation problem we remove " $y_{p1} + y_{p2} \leq 1$ If $p1$ and $p2$ don't have the same base ".

In the ASP formulations we remove the same pairing_base constraint:
:- select_pairing(P1), select_pairing(P2), pairing_base(P1,B1), pairing_base(P2,B2), B1 != B2.

We use the Clingo API in Python to solve the generation problem, then with the generated rosters we solve the assignement problem :

$$\text{Minimize} \quad \sum_{l\in L}\sum_{r\in R} C_l^r \cdot x_l^r \tag{30}$$

$$\text{subject to} \quad \sum_{r\in R} x_{lr} \leq 1 \qquad \forall l \in L \tag{31}$$

$$\sum_{l\in L_u}\sum_{r\in R} a_{rp} \cdot x_{lr} = n_{pu} \qquad \forall p \in P,\ u \in U_p \tag{32}$$

$$x_{lr} \in \{0,1\} \qquad \forall l \in L, r \in R$$

At the end when we get a solution for each subproblem we can construct the solution of the entire master problem.

5.2 Column Generation

In the associated dual problem we don't have the b_l variable anymore :

$$\text{Max} \quad \sum_{l\in L} -v_l + \sum_{p\in P}\sum_{u\in U_p} n_{pu} w_{pu} \tag{33}$$

$$\text{subject to} \quad -v_l + \sum_{p\in P}\sum_{u\in U_p} a_{rp} w_{pu} \leq C_l^r \qquad \forall l \in L_u,\ r \in R \tag{34}$$

$$w_{pu} \in \mathbb{R} \qquad \forall p \in P,\ u \in U_p \tag{35}$$

$$v_l \geq 0 \qquad \forall l \in L \tag{36}$$

$$\tag{37}$$

Then in the ASP pricing problem we remove the **delta** variable and the following constraints :

:- delta(L), member_base(L,B), pairing_base(P,B), select_pairing(P).
:- not delta(L), member_base(L,B1), pairing_base(P,B2), select_pairing(P), B1 != B2.
:- select_pairing(P1), select_pairing(P2), pairing_base(P1,B1), pairing_base(P2,B2), B1 != B2.

And this term from the objective:
-B: select_member(L), delta(L), dual_b(L,B).

f

6 Instances construction

Unfortunately, it was not possible to obtain instances from the industry, and suitable instances for our specific crew rostering problem are not available online. While there are datasets for similar problems, they do not meet our requirements. Therefore, I had to construct the instances myself, making every effort to ensure they are as realistic as possible to facilitate the analysis and testing of the proposed methods.

Pairings Data Structure and construction criteria:
Each pairing in the pairings data file includes the following fields:

1. **pairing_id:** Each pairing is assigned a unique identifier incrementally from 1 up to the total number of pairings.
2. **flight_time:** The flight time for each pairing is randomly selected within a range of 8 to 25 hours.
3. **starting_time:** The start time is randomly generated within the specified range of days in the month (`first_day` to `last_day - 4`) and within a 24-hour period. This ensures that pairings can start at any time of day.
4. **end_time:** The end time is determined by adding the flight time and additional random time (to simulate preparation and post-flight activities) to the start time. The additional random time is calculated as a random value between twice the flight time and two and a half times the flight time, in hours.
5. **starting_minute:** The start time is converted to the total number of minutes from the beginning of the month.
6. **end_minute:** Similarly, the end time is converted to the total number of minutes from the beginning of the month.
7. **working_days:** This field contains a list of the days (as integers) that the pairing spans, based on the start and end times.
8. **base:** The base from which the pairing starts is randomly selected from a list of provided bases.
9. **positions:** A list of positions required for the pairing, including a fixed list of essential positions and potential additional positions. The fixed list includes pilot, copilot, purser, steward, and hostess. To simulate the variety of crew members needed, we add at most two additional members: sometimes none, sometimes one (either a hostess or a steward), and sometimes two (which can be two hostesses, two stewards, or one hostess and one steward). This method ensures variability and realism in crew composition. In the future, it is entirely possible to to integrate optional positions such as technician or engineer.

We can illustrate the construction of the pairings with this:

Algorithm 4 Pairings Construction for `generate_instance`

0: **Input:** `num_pairings`, `month`, `first_day`, `last_day`, `bases`, `positions`
0: **Output:** `pairings`, `base_pairings`, `selected_bases`
0: `pairings` ← []
0: `base_pairings` ← {`base: [] for base in bases`}
0: `selected_bases` ← {`base: False for base in bases`}
0: **for** $i \leftarrow 1$ to `num_pairings` **do**
0: `flight_time` ← random integer between 8 and 25
0: `start_day` ← random integer between `first_day` and `last_day - 4`
0: `start_time` ← random `datetime` in `month`, `start_day`, and hour/minute
0: `end_time` ← `start_time` + `flight_time` + random additional time
0: `working_days` ← list of days from `start_time` to `end_time`
0: `base` ← random choice from `bases`
0: `pairing_positions` ← `positions` + random sample of additional positions
0: Add `i + 1` to `base_pairings[base]`
0: Set `selected_bases[base]` to `True`
0: Add { "pairing_id": $i + 1$, "flight_time": `flight_time`, "starting_time": `start_time`, "end_time": `end_time`, "working_days": `working_days`, "base": `base`, "positions": `pairing_positions` } to `pairings`
0: **end for**=0

Having the pairings data file and the crew members number, the main challenge is to find an efficient criteria to contruct a dataset for the crew members which maximizes the chances to provide a feasable solution for our problem. For this we have experienced several criteria.

Crew Members Data Structure:
Each crew member in the crew members data file includes the following fields:

1. **member_id:** A unique identifier for each crew member.
2. **base:** The base where the crew member is stationed.
3. **position:** The role or position of the crew member (e.g., pilot, co-pilot, hostess, steward).
4. **favorite_pairings:** A list of pairing_ids that the crew member prefers to be assigned to.
5. **favorite_rest_days:** A list of days (as integers) that the crew member prefers to have as rest days.

6.1 First criteria: Totally random construction

To create realistic instances, the first intuitive approach involves generating crew members with random attributes within specified parameters. This naive method ensures a variety of scenarios for initial testing and analysis. Here is a detailed explanation of how each element in the two data files is constructed:

Crew Members Data Construction:

1. **member_id:** Each crew member is assigned a unique identifier incrementally from 1 up to the total number of crew members.
2. **base:** The base where the crew member is stationed is randomly selected from the bases that have been assigned pairings, ensuring all bases with pairings are staffed.
3. **position:** The role or position of the crew member (e.g., pilot, co-pilot, hostess, steward) is randomly chosen from a provided list of positions.
4. **favorite_pairings:** A list of pairing IDs that the crew member prefers, randomly selected from the pairings assigned to their base. The number of favorite pairings is limited by a specified parameter.
5. **favorite_rest_days:** A list of preferred rest days, randomly chosen within the range of days in the month. The number of favorite rest days is also limited by a specified parameter.

Algorithm 5 Crew Members First Construction for `generate_instance`

Input: `pairings`, `num_crew_members`, `base_pairings`, `favorite_pairings_number`, `favorite_rest_days_number`, `last_day`, `first_day`
Output: `crew_members`
`crew_members` ← []
`selected_bases` ← [base for base in `bases` if `selected_bases`[base] is True]
for $i \leftarrow 1$ to `num_crew_members` **do**
  `base` ← random.choice(`selected_bases`)
  `position` ← random.choice(`positions`)
  `favorite_pairings` ←random.sample(`base_pairings`[`base`], min(len(`base_pairings`[`base`]), `favorite_pairings_number`))
  `favorite_rest_days` ←random.sample(range(`last_day`,`first_day`+1),`favorite_rest_days_number`)
  `crew_members.append({ "member_id": i, "base": base, "position": position, "favorite_pairings": favorite_pairings, "favorite_rest_days": favorite_rest_days })`
end for=0

6.2 Second criteria: Weighted Random Construction Based on Pairing Needs

For a more refined approach, the second criterion involves generating crew members based on the distribution of pairings across different bases and positions. This method aims to align the number of crew members with the specific needs identified in the pairings data, thus creating a more balanced and realistic dataset. Here is a detailed explanation of how each element in the two data files is constructed:

Crew Members Data Construction:

1. **member_id:** Each crew member is assigned a unique identifier incrementally from 1 up to the total number of crew members.
2. **base:** The base where the crew member is stationed is determined through a weighted random selection process. The weight for each base-position pair is calculated based on the proportion of pairings that require that specific base and position combination.
3. **position:** Similar to the base assignment, the position of the crew member is determined through a weighted random selection process. This ensures that the number of crew members for each position is proportional to the requirements observed in the pairings data.
4. **favorite_pairings:** A list of pairing IDs that the crew member prefers, randomly selected from the pairings assigned to their base. The number of favorite pairings is limited by a specified parameter.
5. **favorite_rest_days:** A list of preferred rest days, randomly chosen within the range of days in the month. The number of favorite rest days is also limited by a specified parameter.

Weighted Random Selection Process:

1. **Calculation of Weights:**
   - Initialize a counter for each base-position pair to zero.
   - Iterate through all pairings and for each position in the pairing, increment the corresponding base-position counter.
   - Calculate the total count of all base-position pairs.
   - Normalize the counts to obtain the weights by dividing each base-position count by the total count.
2. **Selection of Bases and Positions:**
   - Create a list of base-position pairs.
   - Perform a weighted random selection of base-position pairs for each crew member based on the normalized weights.

This approach ensures that the distribution of crew members across different bases and positions aligns closely with the actual demands observed in the pairings data, leading to a more realistic and practical dataset for further analysis.

Algorithm 6 Crew Members Second Construction for `generate_instance`

Input: `pairings`, `num_crew_members`, `base_pairings`, `favorite_pairings_number`, `favorite_rest_days_number`, `last_day`, `first_day`
Output: `crew_members`
`enum_base_position` ← {(base, position): 0 for base in `bases` for position in `positions`}
`total_base_position` ← 0
`pairings` ← []
`base_pairings` ← {`base: [] for base in bases`}
for $i \leftarrow 1$ to `num_pairings` **do**
  Generate random `flight_time`, `start_time`, and `end_time`
  `working_days` ← list of days from `start_time` to `end_time`
  `base` ← random choice from `bases`
  `pairing_positions` ← `positions` + random sample of ["hostess", "steward"] positions
  Add `i + 1` to `base_pairings[base]`
  for each `position` in `pairing_positions` **do**
    `enum_base_position[base, position]` ← `enum_base_position[base, position]` +1
    `total_base_position` ← `total_base_position` +1
  end for
  Add pairing details to `pairings`
end for
for each `base` in `bases` **do**
  for each `position` in `positions` **do**
    `enum_base_position[base, position]` ← `enum_base_position[base, position]` / `total_base_position`
  end for
end for
`bases_positions` ← list of (base, position) pairs
`sorted_bases_positions` ← random choices from `bases_positions` with weights `enum_base_position` for `num_crew_members`
`crew_members` ← []
for $i \leftarrow 1$ to `num_crew_members` **do**
  `base, position` ← `sorted_bases_positions[i]`
  `favorite_pairings` ← random sample from `base_pairings[base]` with size `favorite_pairings_number`
  `favorite_rest_days` ← random sample from range `first_day` to `last_day` with size `favorite_rest_days_number`
  Add { "member_id": i, "base": `base`, "position": `position`, "favorite_pairings": `favorite_pairings`, "favorite_rest_days": `favorite_rest_days` } to `crew_members`
end for=0

6.3 Third criteria: Base and Position Coverage Before Random Assignment

The third criterion aims to ensure that every base and position is initially covered by at least one crew member before the remaining crew members are assigned based on a weighted random selection. This method helps guarantee that all necessary bases and positions are represented from the start, thus improving the feasibility of the generated dataset. Here is a detailed explanation of how each element in the two data files is constructed:

Crew Members Data Construction:

1. **member_id:** Each crew member is assigned a unique identifier incrementally from 1 up to the total number of crew members.
2. **base:**
   - Initially, assign one crew member to each base-position combination to ensure coverage.
   - For the remaining crew members, the base is determined through a weighted random selection process. The weight for each base-position pair is calculated based on the proportion of pairings that require that specific base and position combination.
3. **position:**
   - Initially, assign one crew member to each base-position combination to ensure coverage.
   - For the remaining crew members, the position is determined through a weighted random selection process, similar to the base assignment.
4. **favorite_pairings:** A list of pairing IDs that the crew member prefers, randomly selected from the pairings assigned to their base. The number of favorite pairings is limited by a specified parameter.
5. **favorite_rest_days:** A list of preferred rest days, randomly chosen within the range of days in the month. The number of favorite rest days is also limited by a specified parameter.

Base and Position Coverage Before Random Assignment:

1. **Initial Coverage:**
   - Create a list of all base-position pairs.
   - Assign one crew member to each base-position pair to ensure every base and position is initially covered.
2. **Calculation of Weights:**
   - Initialize a counter for each base-position pair to zero.
   - Iterate through all pairings and for each position in the pairing, increment the corresponding base-position counter.
   - Calculate the total count of all base-position pairs.
   - Normalize the counts to obtain the weights by dividing each base-position count by the total count.
3. **Selection of Bases and Positions:**

- Perform a weighted random selection of base-position pairs for each remaining crew member based on the normalized weights.

This approach ensures that every base and position is initially covered by at least one crew member, while still reflecting the distribution of needs observed in the pairings data through weighted random selection for the remaining crew members.

Algorithm 7 Crew Members Thirs Construction for `generate_instance`

0: **Input:** `pairings`, `num_crew_members`, `base_pairings`, `favorite_pairings_number`, `favorite_rest_days_number`, `last_day`, `first_day`
0: **Output:** `crew_members`
0: `enum_base_position` ← {(base, position): 0 for base in `bases` for position in `positions`}
0: `total_base_position` ← 0
0: `pairings` ← []
0: `base_pairings` ← {`base: [] for base in bases`}
0: **for** $i \leftarrow 1$ to `num_pairings` **do**
0: Generate random `flight_time`, `start_time`, and `end_time`
0: `working_days` ← list of days from `start_time` to `end_time`
0: `base` ← random choice from `bases`
0: `pairing_positions` ← `positions` + random sample of ["hostess", "steward"] positions
0: Add `i + 1` to `base_pairings[base]`
0: **for** each `position` in `pairing_positions` **do**
0: `enum_base_position[base, position]` ← `enum_base_position[base, position]` +1
0: `total_base_position` ← `total_base_position` +1
0: **end for**
0: Add pairing details to `pairings`
0: **end for**
0: `bases_positions` ← list of (base, position) pairs
0: `crew_members` ← []
0: **for** $i \leftarrow 1$ to length of `bases_positions` **do**
0: `base, position` ← `bases_positions[i]`
0: `favorite_pairings` ← random sample from `base_pairings[base]` with size `favorite_pairings_number`
0: `favorite_rest_days` ← random sample from range `first_day` to `last_day` with size `favorite_rest_days_number`
0: Add { "member_id": i, "base": `base`, "position": `position`, "favorite_pairings": `favorite_pairings`, "favorite_rest_days": `favorite_rest_days` } to `crew_members`
0: **end for**
0: **for** each `base` in `bases` **do**
0: **for** each `position` in `positions` **do**
0: `enum_base_position[base, position]` ← `enum_base_position[base, position]` / `total_base_position`
0: **end for**
0: **end for**
0: `sorted_bases_positions` ← random choices from `bases_positions` with weights `enum_base_position` for `num_crew_members`
0: **for** $i \leftarrow$ length of `bases_positions` + 1*to* `num_crew_members` **do**
0: `base, position` ← `sorted_bases_positions[i]`
0: `favorite_pairings` ← random sample from `base_pairings[base]` with size `favorite_pairings_number`
0: `favorite_rest_days` ← random sample from range `first_day` to `last_day` with size `favorite_rest_days_number`
0: Add { "member_id": i, "base": `base`, "position": `position`, "favorite_pairings": `favorite_pairings`, "favorite_rest_days": `favorite_rest_days` } to `crew_members`
0: **end for**=0

6.4 Fourth criteria: Ensuring Base and Position Diversity with Dynamic Position Allocation

The fourth criterion addresses the need to ensure that each base and position is covered adequately while considering the dynamic nature of position allocation within pairings. In some pairings, multiple crew members can have the same position (e.g., stewards and hostesses). This criterion dynamically adjusts the crew members' allocation to bases and positions based on the pairings data. Here is a detailed explanation of how each element in the two data files is constructed:

Crew Members Data Construction:

1. **member_id:** Each crew member is assigned a unique identifier incrementally from 1 up to the total number of crew members.
2. **base:**
   - Initially, assign crew members to bases based on the selected bases from the pairings data. Only the bases involved in the pairings are considered.
   - For the remaining crew members, the base is determined through a weighted random selection process, ensuring diversity across the selected bases.
3. **position:**
   - Initially, assign positions to crew members based on the most frequently occurring positions in the pairings for each base.
   - For the remaining crew members, positions are determined through a weighted random selection, considering the distribution of positions in the pairings.
4. **favorite_pairings:** A list of pairing IDs that the crew member prefers, randomly selected from the pairings assigned to their base. The number of favorite pairings is limited by a specified parameter.
5. **favorite_rest_days:** A list of preferred rest days, randomly chosen within the range of days in the month. The number of favorite rest days is also limited by a specified parameter.

Base and Position Diversity with Dynamic Position Allocation:

1. **Initial Allocation:**
   - Create a list of all selected base-position pairs from the pairings data.
   - Assign crew members to these base-position pairs initially to ensure coverage.
2. **Calculation of Weights:**
   - Initialize a counter for each base-position pair to zero.
   - Iterate through all pairings and for each position in the pairing, increment the corresponding base-position counter.
   - Calculate the total count of all base-position pairs.
   - Normalize the counts to obtain the weights by dividing each base-position count by the total count.

3. **Dynamic Position Allocation:**

   - For the initial set of crew members, allocate positions based on the most frequently occurring positions in the pairings for each base.
   - For the remaining crew members, perform a weighted random selection of base-position pairs based on the normalized weights.

This approach ensures that each base and position is adequately covered while dynamically adjusting the crew members' allocation based on the actual distribution of positions within the pairings. It allows for the flexibility needed to accommodate pairings where multiple crew members can have the same position, thus maintaining the integrity and realism of the generated dataset. The pseudo-code is similar to the previous criteria's, we just modify the initial coverage of each base positions with the longest found list in the pairings of this base instead of the constant one in the inputs.

6.5 Effectiveness of the Criteria

Based on our experiences, the first and second criteria are not very effective for producing feasible instances. Specifically, these criteria often require a very large number of crew members relative to the number of pairings, and they work primarily with a small number of bases (at most 3), which is cannot be compared to real-world scenarios.

On the other hand, the third criterion, which involves initial coverage, is significantly more effective in generating feasible instances. It is particularly good as it allows for the utilization of all bases within a country, ensuring a more comprehensive and practical solution. The fourth criterion is also quite effective, we can consider it as a small improvement of the 3rd criterion.

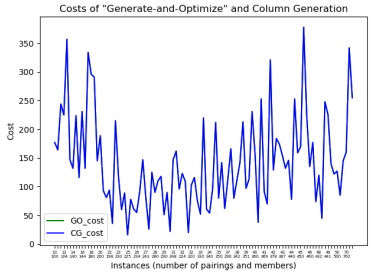
7 Exact methods results

The provided instances where constructed with the last criteria and the results were found with the second approach based (the separation in the bases subproblems).

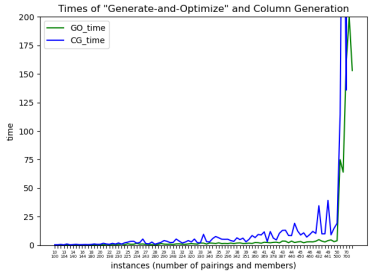
In the results we compare the three methods:

- Generate-and-Optimize (**GO**)
- A second Column Generation (**CG**) with all single rosters (single rosters are rosters with only one pairng) in the initialization (and a dummy variable so that the RMP is feasable in the first iterations).

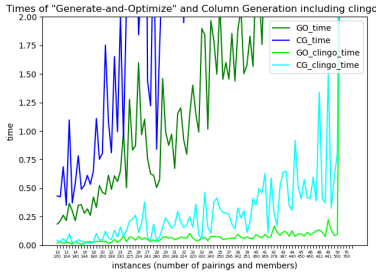
The constructed instances are inspired by the major German airline Lufthansa, which typically has two bases for their crew rostering problem. Accordingly, we fixed the number of bases to two in our instances. The rostering period depends on the number of pairings: 7 days if there are no more than 50 pairings, 14 days if there are between 51 and 500 pairings, and 28 days if there are more than 500 pairings. In total, we tested 99 instances, with sizes ranging from 10 pairings to 100, and up to 80 pairings for 800 pairings (we stopped at 80 pairings because the Column Generation was too slow with 2 bases).



(a) Cost results of the 'GO' and 'CG' algorithms



(b) Times results of the 'GO' and 'CG' algorithms



(c) Times results of the 'GO' and 'CG' algorithms with clingo time

Figure 1: Exact methods

Next, we attempted to implement a new Branch-and-Cut-and-Price algorithm using the **Coluna.jl** framework. We used the following formulation, which is equivalent to our previous one:

$$\text{Minimize} \quad \sum_{l \in L} \sum_{\substack{p \in P \\ base(l)=base(p) \\ p \text{ not in favorite\_pairings}(l)}} w(l) \cdot \frac{f_p}{4} \cdot y_{lp} \quad + \quad \sum_{l \in L} \sum_{\substack{d \in Days \\ d \text{ in favorite\_days}(l)}} w(l) \cdot (1 - z_{ld}) \tag{38}$$

$$\text{subject to} \quad \sum_{\substack{l \in L_u \\ base(l)=base(p)}} y_{lp} = n_{pu} \qquad \forall p \in P,\ u \in U_p \tag{39}$$

$$\sum_{\substack{p \in P \\ base(l)=base(p)}} f_p \cdot y_{lp} \leq F_{max} \qquad \forall l \in L \tag{40}$$

$$\sum_{\substack{p \in P \\ base(l)=base(p)}} y_{lp} \leq P_{max} \qquad \forall l \in L \tag{41}$$

$$\sum_{d \in D} z_{ld} \geq Off_{min} \qquad \forall l \in L \tag{42}$$

$$y_{lp1} + y_{lp2} \leq 1 \qquad \forall l \in L,\ \text{If rest time between } p1 \text{ and } p2 \text{ is} < R_{min} \tag{43}$$

$$y_{lp1} + y_{lp2} \leq 1 \qquad \forall l \in L,\ \text{If } p1 \text{ and } p2 \text{ don't have the same base} \tag{44}$$

$$\sum_{d \in CD} z_{ld} \geq 1 \qquad \forall l \in L,\ \forall \text{ set of consecutive days } CD \text{ s.t } |\text{CD}| = \text{CW} \tag{45}$$

$$y_{lp} + z_{ld} + \leq 1 \qquad \forall l \in L,\ \text{If } d \text{ is a working day for } p \tag{46}$$

$$z_{ld} \leq 1 - \frac{\sum_{\substack{p \in P \\ d \text{ is a working days for } p}} x_{lp}}{|P|} \qquad \forall l \in L,\ \forall d \in Days \tag{47}$$

$$z_{ld} \geq 1 - \sum_{\substack{p \in P \\ d \text{ is a working days for } p}} x_{lp} \qquad \forall l \in L,\ \forall d \in Days \tag{48}$$

$$y_{lp} \in \{0, 1\} \qquad \forall l \in L,\ \forall p \in P,\ \text{if base(l)=base(p)} \tag{49}$$

$$z_{ld} \in \{0, 1\} \qquad \forall l \in L,\ \forall d \in D \tag{50}$$

We were able to solve large instances in Julia using the CPLEX solver, but unfortunately, we couldn't solve the Branch-and-Cut-and-Price problem using the Coluna framework due to excessively slow resolution times.

8 greedy heuristic and simulated annealing

We observed that the resolution time of our exact methods can increase significantly, especially with a small number of bases like Lufthansa's. It is very unlikely that these algorithms can solve real airline company instances, where a crew rostering problem might involve around 1000 pairings over a one-month period. Therefore, we are now searching for alternative methods that can solve our problem much faster. The first ones we explored are a greedy heuristic and a metaheuristic based on simulated annealing.

8.1 Greedy heuristic

For the greedy heuristic method, the main idea is to select a pairing p in each iteration and assign its crew members who are available, while maximizing the number of crew members who have p as one of their favorite pairings.

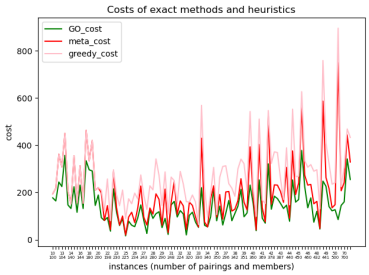
Algorithm 8 Greedy heuristic

```
1: Input: pairings, members, bounds
2: Output: pairings_assignement, members_assignement, cost
   pairings_assignement = dict() members_assignement = dict() cost = 0
3: for p in pairings do
4:    crew = available_best_crew(pairings_assignement, members_assignement, bounds)
5:    pairings_assignement[p] = crew
6:    for l in crew do
7:       members_assignement[l] += [p]
8:    end for
      cost += compute_cost(pairing, crew)
9: end for
10: return  pairings_assignement, members_assignement, cost =0
```

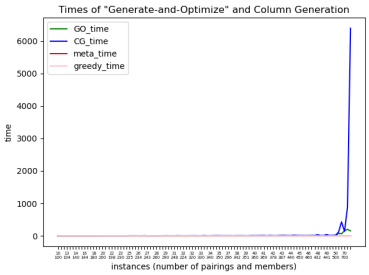
8.2 Simulated annealing

For simulated annealing, we first need to define a neighborhood structure. The initial approach is to randomly select a pairing and randomly change one of its current members. However, this approach does not yield efficient results, as there is often little difference between neighboring solutions when only one crew member is changed. To improve this, we implemented a similar neighborhood structure, but instead of changing just one crew member, we change as many as possible to achieve significant differences between neighbors. The results were significantly better than the first approach, even when we reduced the parameters for this approach (initial temperature, number of iterations, alpha). We stop the algorithm as soon as the temperature reaches a fixed limit.

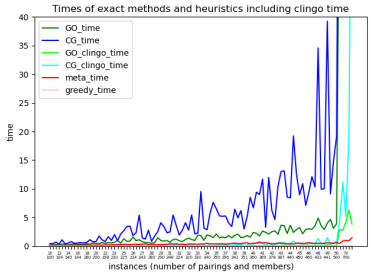
The greedy heuristic and simulated annealing were implemented in Python to compare their execution times with the 'GO' and 'CG' algorithms. For the 99 previously tested instances, here is a comparison between the exact methods and the heuristics:



(a) Cost results of the exact methods, greedy and SA



(b) Times results of the 'GO', 'CG', greedy and SA algorithms

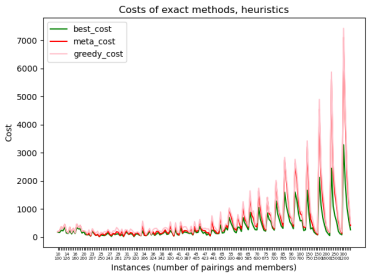


(c) Times results of the 'GO', 'CG', greedy and SA including clingo time

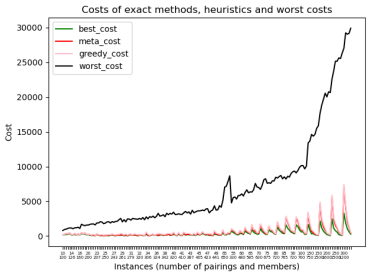
Figure 2: Exact methods and heuristics

As we can see, the greedy heuristic and simulated annealing are significantly faster than the exact methods, even outperforming the roster generation by Clingo. However, the instances solved so far are relatively small compared to real-world scenarios, especially those faced by large airline companies. Therefore, to better assess the efficiency of our heuristics on larger instances, we plan to use the latest formulation of our problem in Julia to find the optimal solution within a reasonable timeframe.

We were able to solve instances with up to 300 pairings and 2,400 members using the latest model in Julia. Below are the results for the cost:



(a) Cost results of the exact methods, greedy and SA



(b) Cost results of the exact methods, greedy and SA with the worst cost

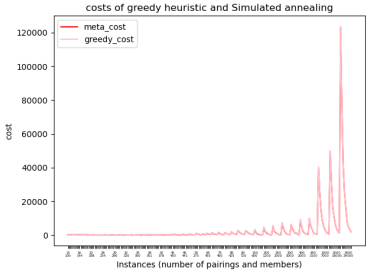
(c) "Efficiency" rates of the greedy and SA

Figure 3: Heuristics efficiency

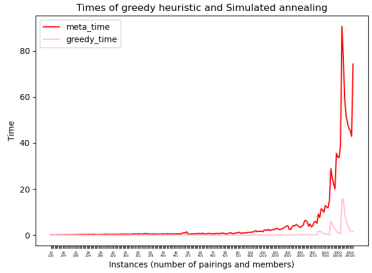
In the first graph, we present the optimal cost along with the costs obtained from the greedy and simulated annealing algorithms. The second graph displays the cost of the worst solution in black (achieved by maximizing the objective). The final graph illustrates the rates of each cost from the greedy and simulated annealing methods, calculated using the following formula to have an indication of how close we are to the optimal cost.:

$$\frac{worst\_cost - solution\_cost}{worst\_cost - best\_cost}$$

We were unable to solve instances with more than 300 pairings in Julia for further analysis; however, we successfully handled much larger instances using our heuristics in Python. Below are the costs and resolution times for up to 3,000 pairings and 30,000 members:



(a) Cost results of the greedy and SA



(b) Cost results of the greedy and SA with the worst cost

Figure 4: Heuristics costs and times

References

[1] Ralf Borndörfer, Uwe Schelten, Thomas Schlechte, and Steffen Weider. *A Column Generation Approach to Airline Crew Scheduling*. Springer Berlin Heidelberg.

[2] T. Breugem, B.T.C. van Rossum, T. Dollevoet, and D. Huisman. *A column generation approach for the integrated crew re-planning problem*, volume 107. Elsevier BV, February 2022.

[3] Thomas Breugem, Twan Dollevoet, and Dennis Huisman. *Analyzing a Family of Formulations for Cyclic Crew Rostering*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

[4] Alberto Caprara, Paolo Toth, Daniele Vigo, and Matteo Fischetti. *Modeling and Solving the Crew Rostering Problem*, volume 46. Institute for Operations Research and the Management Sciences (INFORMS), December 1998.

[5] Muhammet Deveci and Nihan undefinedetin Demirel. *A survey of the literature on airline crew scheduling*, volume 74. Elsevier BV, September 2018.

[6] Walid El Moudani, Carlos Alberto Nunes Cosenza, Marc de Coligny, and Félix Mora-Camino. *A Bi-Criterion Approach for the Airlines Crew Rostering Problem*, page 486–500. Springer Berlin Heidelberg, 2001.

[7] Torsten Fahle, Ulrich Junker, Stefan E. Karisch, Niklas Kohl, Meinolf Sellmann, and Bo Vaaben. volume 8. Springer Science and Business Media LLC, 2002.

[8] Atoosa Kasirzadeh, Mohammed Saddoune, and François Soumis. *Airline crew scheduling: models, algorithms, and data sets*, volume 6. Elsevier BV, June 2017.

[9] Niklas Kohl and Stefan E. Karisch. *Airline Crew Rostering: Problem Types, Modeling, and Optimization*, volume 127. Springer Science and Business Media LLC, March 2004.

[10] Axel Parmentier and Frédéric Meunier. *Aircraft routing and crew pairing: Updated algorithms at Air France*, volume 93. Elsevier BV, June 2020.

[11] Frédéric Quesnel, Alice Wu, Guy Desaulniers, and François Soumis. *Deep-learning-based partial pricing in a branch-and-price algorithm for personalized crew rostering*, volume 138. Elsevier BV, February 2022.

[12] Yifan Xu, Sebastian Wandelt, and Xiaoqian Sun. *Airline scheduling optimization: literature review and a discussion of modelling methodologies*, volume 3. Oxford University Press (OUP), December 2023.