Vrije Universiteit Amsterdam

KLM Royal Dutch Airlines

Master Thesis

Optimization of a complex Holiday Assignment Model

Author: Alexandros Tsiamas (2739450)

1st supervisor:	Ger Koole	
daily supervisor:	Marc Paelinck	(KLM)
2nd reader:	Eduard Belitser	

A thesis submitted in fulfillment of the requirements for
VU Master of Science degree in Business Analytics (Optimization of Business Processes)

November 22, 2023

"If you add sorrows and happiness, the sum becomes the human life.
If you subtract happiness from sorrows, the net result is the living world.
We divide our joys, but not our sorrows.
Sorrows always multiply, joys seldom add. "
by Ravi Panamanna

Abstract

Context. This research focuses on enhancing a Holiday Assignment Model. First, we start by introducing the problem with its description, its relevance to the company and the corresponding challenges. Then, there is a detailed representation of the Integer Linear Programming Problem formulation, emphasizing and explaining in detail the objective function, constraints and variables. Special attention is given to the capacity and conflict constraints, which have the highest impact on the performance of the model, as well as to the crucial trade-off between penalties for violations and rewards for assignments. Then, several model improvement techniques are explored and results are presented using tables, visualizations and justifications.

Goal. The primary goal of this research is to determine if it is possible to improve the performance of the Holiday Assignment Model in terms of runtime by employing some well-known techniques, but also custom ones, giving planners more time and flexibility to experiment with different scenarios before publishing the holiday schedule.

Method. This project involves two main parts: formulating the ILP model and applying optimization techniques for improvement. In the first part, a reference MIP model was created, similar to the one that is used in the current assignment application. In the second part, different techniques are investigated to improve the model's performance and reduce the number of conflicts produced by the soft constraints. Well-known strategies are implemented, such as Column Generation, Lazy Constraints, Warm Start and Parameter Tuning, but also customized approaches such as Multi-objective optimization and several weekly capacity relaxation cases instead of daily ones.

Results. The results focus basically on the Senior Purser rank Cabin Crew, but the model is tested for all ranks in the first part. Although no substantial improvement of the model was obtained, near optimality can be reached within a runtime that is acceptable for the users of the model. However, due to

the use of soft constraints, some capacity and conflict violations are allowed intentionally in order to avoid infeasibility, which would not give the planners any information about how to modify the model inputs. A second reason for adding slack variables is to facilitate the search strategy of the solver so it can look for solutions in the relaxation of the constraints. However, the second part's results reveal that despite the employment of numerous optimization techniques, no significant improvement in performance is achieved. Even if the Final Gap is lower, there are still a lot of violations that require manual fixing by the planners to test different scenarios.

Conclusions. These results suggest that improving performance and achieving better solutions within a reasonable time is challenging. Nevertheless, a lot of insights arise from the outcome, highlighting that there is a crucial issue with the given capacity, leading to a lot of unavoidable violations in both capacity and conflict constraints. On top of that, Collective Labour Agreement rules have a very high impact on this, making the model very complex. Even if the problem gets reformulated in terms of the decision variables, the complexity persists and an optimal solution within a reasonable time remains elusive.

Acknowledgements

My profound thanks are reserved for Vrije Universiteit Amsterdam. I am honored to acknowledge the contributions of Ger Koole through the meetings we had on a regular basis. Furthermore, special thanks to Annemieke van Goor, the Business Analytics internship coordinator for being in contact with me and replying as fast as possible to all my questions and especially for being the connecting pattern between me and KLM in order to find the corresponding thesis internship position. Additionally, big thanks to Eduard Belitser for being my second reader.

I would like to express my deepest appreciation to KLM, the Data & Technology Department and the Business Platform Flight. Being a member of the OnTrack team, I am grateful for the mentorship and advice provided by my supervisor Marc Paelinck (leader of the team) who gave me the opportunity to do my research on such a challenging subject. I could not have asked for a more exceptional supervisor, not only for his unwavering dedication to the success of my thesis but also for his profound expertise in the realms of Mathematics and Data Science. Our meetings evolved also into productive brainstorming sessions, and undoubtedly, he left his mark on both my thesis and my initial steps into the professional world. I extend my thanks to the VakPlan team (Caoimhe Ni Mhaonaigh, Tomas Pothaczky, and Robert Hillen) who supported me throughout this journey. KLM's agile way of working and scrum methodology played a significant role in my exponential progress towards my goal. Furthermore, big thanks to Kunal Kumar, Marjan van den Akker, Tim Lamballais Tessensohn and Wiro Keijer for the informative meetings we had.

Last, I would like to express my heartfelt gratitude to my family, who has consistently stood by my side through every challenge I have faced in life. First I would like to thank my parents, Babis and Rika, for supporting me before and during my master's thesis. Last, I would be remiss if not mentioning my brother, John. Without his support, encouragement and tips, this thesis would not have been possible.

Contents

List of Figures

List of Tables

LIST OF TABLES

List of Algorithms

1

Introduction

1.1 Problem Description

Airline companies have many internal concerns and problems to deal with in order to maximize profit, such as the pricing of the tickets, load of flights, and appropriate amount of catering. The five focus aspects of KLM are: Sustainability, Customer, Employee, Operations and Finance. Based on the `Employee` part, a very important internal organizational process is the holiday assignment of the cabin crew. KLM is such a big company that it needs an efficient model to plan the vacation of the Flight Attendants (FAs) in a fair way.

The current assignment model was written in Java, needed a lot of manual corrections and could not reach an optimal solution within a reasonable time. The complexity of the problem arises from the common obvious preference of most people having holidays in certain months (Summer months or Christmas), but in the airline industry, it contrasts with the demand from the customers who also travel a lot during those months. Moreover, the importance comes from KLM's focus on employee satisfaction and the fact that crew members have irregular rosters that limit the possibilities to manage their own free time. The latter is one of the main factors that influence employee satisfaction. Due to that, the company uses a holiday assignment model that takes crew requests into account and the essence of the challenge consists of formulating a model that maximizes the number of those granted requests while respecting the numerous sometimes complex CLA (Collective Labour Agreement: collective agreements between an employer and their employees about salaries, working conditions and employee benefits) rules and the holiday capacity. Furthermore, there is a crucial trade-off involved in the model, between the capacity violations and the conflicts, which has the highest impact on the optimization process. Extending

that, the model must aim to reach a valid and (nearly) optimal solution within a reasonable time (model improvement), which would give manpower planners more time to try out several scenarios before publishing the final holiday assignments. The holiday assignment process is too short to wait for this optimal solution, so the planners use a sub-optimal solution which contains some constraint violations due to the use of soft constraints. These violations need to be fixed manually by the planners. On top of that, the use of conflict constraints, in combination with the very tight given capacity, makes the runtime even worse.

The first part of this research involves implementing the model in Python using Gurobi, based on the Mathematical Model described in Section 3 along with an explanation of the constraints and the constants employed. The second part addresses the research question, focusing on optimizing the model by employing different techniques to enhance performance and reduce computational time.

1.2 Research Question

Based on the description of the problem, our research question is formulated as follows:

Can we define ways to improve the efficiency and performance of the cabin crew's holiday assignment model for KLM?

1.3 Relevance of the problem

Planning the vacation of the FAs in a fair way is also related to the revenue of KLM. The company wants to increase the satisfaction of the FAs, by providing a fair schedule of holidays, because they are the ones who provide service to the customers. Thus it is very crucial to have happy employees. If we elaborate more on this, a fair holiday assignment model can make the employees stay for much longer in the company, which means that the company will also avoid extra training costs for the new employees that would replace them. The job of Flight Attendants is complicated in terms of free time, since some of them might be away from their home and their family for more than 24 hours in a row. That is why they appreciate a lot their holidays and this has a great impact on their satisfaction. Crew members have relatively limited influence on their roster, and thus also on the planning of their days off. KLM totally understands this situation and that is why lots of request options are given to the Flight Attendants, but at the same time it must

be fair between them. In general, it is very clear in all companies that if the employees are dissatisfied, then there is a significant difference in performance and development. It is very easy to maximize the happiness of some employees, but at the same time there might be a decrease of it in the rest, which means that is all about balancing and satisfying at the same level all of them. That can be achieved through fairness and the holiday assignment model aims towards that direction in KLM.

1.4 Motivation

This project was more than challenging from the first time considering the complexity of the model and its constraints. The outcome of the Java model that needs a lot of manual corrections provided some good and bad conclusions. The good part was that with some manual corrections, everything could be fixed, on the other hand, there was a cross-question "Why is it that easy to fix it manually and Gurobi cannot do it?". It was known that people before tried to answer this question for the corresponding assignment model of KLM but unsuccessfully, thus dealing with that project gave a feeling of a real challenge. Working on this project meant tackling the real-world task of creating a holiday assignment model for the company and exploring new optimization techniques that had not been tried before. Even if the outcome turns out to be negative, it will still provide valuable insights into why the model cannot be further improved. My motivation for this project stemmed not only from its demanding nature but also from the potential benefits it could bring to the company, impacting the level of crew's satisfaction.

1.5 General Information

KLM is the oldest operating airline in the world, still operating under its original name. The organization "D&T" stands for Data & Technology and is responsible for developing and maintaining software applications. D&T's Business Flight Platform operates for the Flight domain. This consists of all the processes directly related to a flight operation: the product on board (such as scheduling, assigning and assisting the flight crew and providing catering on board) and all flight technical aspects (e.g. creating flight plans and processing the communication between the aircraft and the Operations Control Department).

The Cabin Crew Planning and Assignment process supports the Cabin Crew Holiday Assignment. This is done with a holiday assignment model called **VakPlan** which has

to satisfy as many requests as possible while respecting a threshold which corresponds to the total vacation capacity per day. The given capacity is determined by the Manpower Planners on a weekly basis, which is derived on one hand from the expected total available crew capacity, and on the other hand from the capacity needed to execute the flight schedule and the premises (e.g. for illness and reserve coverage). The company has in total around 7,000 FAs who are split up into 4 different ranks based on their experience: SP (Senior Pursers), BC (Business Class FA), CA (Cabin Attendant), and PU (Pursers). The Holiday Assignment process is performed twice a year using Vakplan and the Summer period is from May to September while the Winter one is from October to April. Additionally, each period is divided into four equal blocks A, B, C and D and each year crew members are designated to a different block ("standard block"). The rotation of the standard blocks is illustrated in Figure 1.1. This happens in order to achieve fairness among all the employees, since some months are always more popular than others for vacation. The block of each FA is given in the input data, so the block rotation is not that relevant to the model's implementation, it is just mentioned in order to explain what is the logic of assigning employees to blocks.

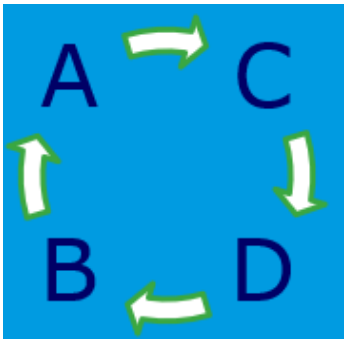

Figure 1.1: Crew's block rotation

The crew members can request a maximum of 3 dates in their standard block. Next to this, they can either request 3 additional dates anywhere inside or outside their standard block, or request to be assigned in another block, called `preferred block`, but without giving any specific start date. Below are the terms for each one:

- **Standard block** dates: All the dates in the standard block, that an FA is assigned.

- **Standard request** dates: Requested dates in the standard block, that an FA is assigned.
- **Preferred block** dates: Preferred dates or dates that lie within the preferred block that an FA has requested.

A request is considered to be granted if and only if the starting date is assigned in a range of 3 days around the requested date or the starting date is within the requested block, thus we say that this day is a **hit**. Furthermore, the model also takes a **priority counter** into account. Each crew member has a priority counter that keeps track of the number of consecutive seasons in which they did not get any preferred request granted. The default value for new crew members is 10, being the lowest priority. If a request is not granted, the priority counter for that crew member is reduced by 1. However, when a request is granted, it is reset to its default value of 10. Thus, lower value means higher priority and the company has employed this approach to ensure fairness, preventing any FA from having their preferred requests declined more often compared to others.

1.6 Solver and Data

The model is currently being refactored, and Python is being used because it is the standard programming language for Data Scientists. Thus, for the implementation of the ILP model and its further optimization, we used Python. Gurobi [1] was used as a solver since it is the standard LP solver at Air France - KLM and it is one of the most powerful solvers when dealing with large-scale optimization problems. The experiments did not run locally, but in the server (cluster) of KLM.

Although the model runs for all the different ranks, the optimization techniques were tested only for the SP rank because it includes the least number of employees and we needed a smaller data sample to avoid very long running times. We were provided data for the Summer period of 2021 to test the model performance. Data includes the rank and block of each employee, their requested standard dates and preferred dates, as well as block ranges and additional information for excluded dates (such as pregnancy reasons). The capacity of employees who are on holiday each day and of employees who start holidays each day is given by the planners and obviously, it is different for each rank. Currently, the data is given in Excel or CSV files and everything is stored afterwards in JSON files [5] because of the preprocessing.

More specifically, 6 different preprocessing files were created in order to deal with the input data from the Excel and CSV files.

- **Blocks preprocessing.** This takes the file with the information about blocks, which shows the starting and ending dates of the summer or winter blocks and returns a JSON file with the range of dates for each of the blocks A, B, C and D.
- **Crew preprocessing.** This is related to all the information about the crew. It outputs the crew member with their standard block, their rank, the part-time percentage (PCT) and the JV days (this is explained in Section 3).
- **Excluded preprocessing.** This works out the corresponding information regarding the excluded dates of each crew member. These are dates that already contain assignments.
- **Capacity preprocessing.** Here two JSON files are created, one for the capacity of people who can be on holidays each day and one with the maximum number of employees who can start their holidays on each day.
- **Diva preprocessing.**. `Diva` is the system in which crew members input their preferences (requests) for the starting dates of their vacation. Thus, this part deals with this information and returns all the allowed starting days for each employee while splitting them into standard block, standard request and preferred request data sets.
- **Final preprocessing.**. The final part of preprocessing is related to sorting out all the information together, by removing the excluded dates from the allowed starting days for each employee and attaching corresponding weights to each type of request.

2

Literature Review

In this section we will discuss related work and do a literature review. We present the closest possible cases, as well as a review of the methods we used for the improvement of the model, referring to the way they are applied and considering the quality of the researchers' results.

2.1 (Non)similar cases

The description of the problem in combination with the complex constraints makes this case non-similar to any other. This distinction primarily arises due to the delicate balance between granting requests, rewards, and penalties for violating constraints, as well as the crew's options for requests. However, the nearest case that can be found is the Nurse Rostering problem as outlined in [7]. Based on that, Qu, R. and He, F. [26] described a constraint programming approach for the decomposition of the Nurse Rostering problem in sub-problems. Moreover, Kharraziha et al. [19] revealed an almost identical large-scale rostering problem encountered by airline and railway companies with highly specific rules and constraints. On top of that, Saemi et al. [27] underlined that crew scheduling problems consist of crew pairing and crew rostering, suggesting a novel mathematical model that optimizes the formulation of both sub-problems combined.

2.2 Soft Constraints

Soft constraints and slack variables are an approach with the purpose of allowing some constraint violations and increasing the feasible region for insightful suggestions. In our research, this is made in order to gain insights into adjusting capacity levels, given the

planners' openness to recommendations in this regard. According to the Model Predictive Control of Kerrigan, E. C. and Maciejowski, J. M. [18], slack variables were used to relax the constraints and corresponding penalties in the objective function in order to avoid infeasibility and the penalties kept the violations as low as possible. A noteworthy research has been made by Smith et al. [30] on penalty functions and solutions. In this context, those solutions violated the constraints, which have been defined on purpose as soft constraints instead of strict ones, allowing them to be regarded as feasible solutions.

2.3 Column Generation

Willhelm et al. [32] showcased the impact of column generation by illustrating its applicability across various problem types. Brunner et al. [6] proposed a technique for scheduling different physicians in hospitals in a model that encompassed not only the demand considerations, but capacity constraints related to maximum shift length, while there was also a fixed cost for each employee. He succeeded in using column generation in combination with some sort of capacity constraints because it was easy to define the columns and find the most promising variables, in contrast with our case. Column Generation is used a lot in the healthcare sector, as highlighted by Shao et al. [29], providing an overview of its main points and steps for an application in patient scheduling problem using Gurobi. One of the most common applications of this method is the cutting stock problem [17], which Pedroso et al. [25] described in detail in terms of problem reformulation to enhance model performance. On the other hand, Muts et al. [22] showed an innovative approach using a non-linear column generation method which involves resource-constrained reformulation and convex relaxation, offering an alternative perspective on this powerful optimization technique.

2.4 Lazy Constraints

Among the optimization techniques that can be applied to our model, Lazy Constraints 4.1.1 stands out as another valuable approach with solid research backing. Pearce et al. [24] underlined the combination of Column Generation and Lazy Constraints to optimize the Liner Ship Fleet Repositioning model. Initially, Column Generation is applied in order to figure out the most crucial variables and avoid dealing with the whole set since it is a Large Scale Problem. Subsequently, Lazy Constraints were introduced in the generated sub-problems (pricing problems) to cut off unnecessary constraints, enhancing efficiency.

Additionally, in a study conducted again by Pearce, R. H. [23] straightforward examples of lazy constraints are provided, showing off their utility. Before mentioning the application of Liner Ship Fleet Repositioning, the strengths of this specific technique are thoroughly analyzed and the term "lazy fashion" is introduced which was applied to the relaxed formulation of the problem where its non-feasible solutions are improved by the addition of extra constraints. Last, Viegas et al. [31] underscored the significance of lazy constraints by presenting a compelling case study for the improvement of a state-of-art graph constraint solver. A comparison between the outcomes of the initial problem formulation and the application of lazy constraints gives out a notable enhancement in efficiency, which can be directly attributed to laziness.

2.5 Warm Start

Occasionally, providing an initial solution to the solver yields superior results and faster convergence in the optimization process. Anbil et al. [2] in their exploration of relaxation with column generation for the airline crew pairing problem, highlighted that the initial step involves generating an initial solution, which can be obtained by solving a similar version of the original problem. Klabjan, D. [20] emphasized the general usefulness of this approach for both column generation and dual variables. Furthermore, Yildirim, E. A. and Wright, S. J. [33] pointed out the significance of warm start strategies when addressing linear programming problems. Throughout their research, they underline that this technique usually reduces complexity and facilitates the derivation of a starting point. Based on the impact of warm-start strategies, Freund, R. M. [13] proved that even if the initial solution is infeasible for the problem, it can be close to feasibility and optimality, ultimately aiming to improve runtime.

2.6 Multiple Objective Functions

Using multiple objective functions in complex problems is a strategy to enhance model optimization and narrow the final optimality gaps. Benayoun et al. [4] designated different categories of problems that their objective function can be decomposed into multiple ones, allowing for a step-by-step pursuit of optimality. However, Gennert et al. [14] highlighted the importance of determining the priority and assigning appropriate weights to the chosen multiple objective functions. Balancing these objectives effectively is key to achieving meaningful and balanced optimization outcomes.

2.7 Parameter and Settings

Gurobi offers a wealth of parameters that can be changed to influence the solver's path towards finding an optimal solution. De Givry, S. and Katsirelos G. [9] mentioned in their research that several cut methods can be employed to impact the linear relaxation of the model. They proved that Clique cuts, in particular, have the effect of optimizing the dual bounds, on which the branch and bound algorithm is based. Another category of cut technique is Gomory cuts [15]. Cornuéjols, G. [8] conducted a literature review in his paper, noting that previous applications of this technique did not lead to optimization improvements. In contrast to these prior findings, he showed that Gomory cuts improved his case and justified it by mentioning the 3 key reasons for their effectiveness. Moreover, Salehispour, A. [28] embarked on a quest for optimality in his model for the aircraft landing problem by using specific values for the parameters TimeLimit but also MIPFocus. The default value of the second one is 0, but he explored using 2 to prioritize finding the optimal solution, while 1 was omitted since it emphasizes the solver's focus on feasibility.

3

Current ILP Model

This section includes all the necessary information about the mathematical formulation of VakPlan. The objective function, constraints and variables are defined and explained, as long as detailed results of all different ranks are illustrated with Figures and Tables.

3.1 ILP

Vakplan can be easily described as an optimization problem which is based on a mathematical formulation, including the objective function, constraints and corresponding decision variables. Below we provide the detailed model:

$$
\begin{aligned}
\max \quad & (W_{hit} + W_1) \sum_{i \in C, j \in V_i} d_{i,j} + (W_{hit} + W_2) \sum_{i \in C, j \in SR_i} d_{i,j} + (W_{hit} + W_3) \sum_{i \in C, j \in PR_i} d_{i,j} \\
& - W_{cu_1} \sum_{j \in D} cu_{1,j} - W_{cu_2} \sum_{j \in D} cu_{2,j} - W_{cu_3} \sum_{j \in D} cu_{3,j} - W_{cu_4} \sum_{j \in D} cu_{4,j} - W_{cu_5} \sum_{j \in D} cu_{5,j} \\
& - W_4 \sum_{i \in C} su_i - W_5 \sum_{i \in C} \alpha u_i - W_6 \sum_{i \in C} pu_i
\end{aligned}
$$

s. t.

$$d_{i,j} \in 0,1 \qquad \forall i,j \in H$$

$$\sum_{j \in D} d_{i,j} \leq 1 \tag{1}$$

$$\sum_{i \in C} d_{i,j} \leq B_j \tag{2}$$

$$\sum_{j \in periodk} \sum_{i} d_{i,j} \geq T * BL_k \tag{3}$$

$$\sum_{j \in periodk} \sum_{i} d_{i,j} \leq T * BH_k \tag{4}$$

$$\sum_{i} \sum_{k=j-L+1}^{j-RV} PCT_i d_{i,k} \leq C_j + cu_{1,j} + cu_{2,j} + cu_{3,j} + cu_{4,j} + cu_{5,j} \tag{5}$$

$$CS_c \sum_{j \in SR_c} d_{c,j} - \sum_{i \in C, j \in D} KS_{c,i,j} d_{i,j} + su_c \geq 0 \qquad \forall c \in C \tag{6}$$

$$CA_c \sum_{j \in SR_c} d_{c,j} - \sum_{i \in C, j \in D} KA_{c,i,j} d_{i,j} + \alpha u_c \geq 0 \qquad \forall c \in C \tag{7}$$

$$CP_c \sum_{j \in PR_c} d_{c,j} - \sum_{i \in C, j \in D} KP_{c,i,j} d_{i,j} + pu_c \geq 0 \qquad \forall c \in C \tag{8}$$

where:

W_{hit} **constant**: reward for assigning an employee to a day that is a hit.

W_1, W_2, W_3 **constants**: weights for each type of possible starting date.

W_{cu_1}, W_{cu_2}, W_{cu_3}, W_{cu_4}, W_{cu_5} **constants**: penalty coefficients for capacity violations.

$d_{i,j}$ **decision variable**: 1 if the vacation of FA i starts on day j, else 0.

H **set**: includes all combinations of $i \in C$ and $j \in D$ that produce a hit

C **set**: includes all the employees

D **set**: includes all the dates

V_c **set**: includes all the standard block dates of each employee $c \in C$

SR_c **set**: includes all the standard requested dates of each employee $c \in C$

PR_c **set**: includes all the preferred requested dates of each employee $c \in C$

$cu_{1,j}$, $cu_{2,j}$, $cu_{3,j}$, $cu_{4,j}$, $cu_{5,j}$ **variable**: surplus variables for capacity violations for each day j.

su_i, αu_i, pu_i **variable**: surplus variables for conflicts violations with crew i.

B_j **constant**: maximum number of starting holidays on day j.

C_j **constant**: total available capacity (FTE) for vacations on day j.

BL_k **constant**: minimum fraction of starting holidays in block k.
BH_k **constant**: maximum fraction starting holidays in block k.
RV **constant**: empty roster days that are added at the start of the holiday and that are not included in the total vacation capacity.
PCT_i **constant**: contract percentage of crew member i with $0 \leq PCT_i \leq 1$
CS_c **constant**, big-M: count of crew with at least one potential preference date conflict with a standard date of crew c.
$KS_{c,i,j}$ **indicator**: 1 if the date causes a standard-preference conflict for day j of crew c, else 0.
CA_c **constant**, big-M: count of crew with at least one potential standard-any conflict with a standard date of crew c.
$KA_{c,i,j}$ **indicator**: 1 if the date causes a standard any conflict for day j of crew c, else 0.
CP_c **constant**, big-M: count of crew with at least one potential priority conflict with a preferred date of crew c.
$KP_{c,i,j}$ **indicator**: 1 if the date causes a priority conflict for day j for crew c, else 0.

(1) - (5) are considered to be the capacity constraints. Constraints (1) make sure that there is a maximum of one assigned vacation per crew member. We do not force the assignment of exactly one vacation to each crew member in order to avoid facing infeasibility, which could otherwise occur if the allocated holiday capacity is insufficient. On top of that, for feasibility reasons we use the constant W_{hit} in order to reward assigning crew members (or penalizing unassigned ones). The purpose of constraints (2) is the equal spread of the starting dates, since this way we set a maximum number of starting holidays per day. Constraints (3) and (4) refer to the corresponding minimum and maximum starting holidays per block and T is the number of vacations that must be assigned, but currently these constraints are off since we want to make the model as light as possible and it was proven that they are already satisfied no matter what. The last capacity constraints (5) take care of the limitation regarding the maximum number of people that can be on holiday each day. Here the second summation iterates over all possible start dates that cause the vacation to overlap day j, while L is the vacation length and RV is the number of RV days. These are empty roster days that are added at the start of the holiday for safety reasons and that are not included in the total vacation capacity. However, RV days are considered as a part of the vacation which means that the total duration of a vacation is equal to $JV + RV$, as illustrated in Figure 3.1. Last, (6) - (8) are the conflict constraints and in the category of the well-known big-M constraints.

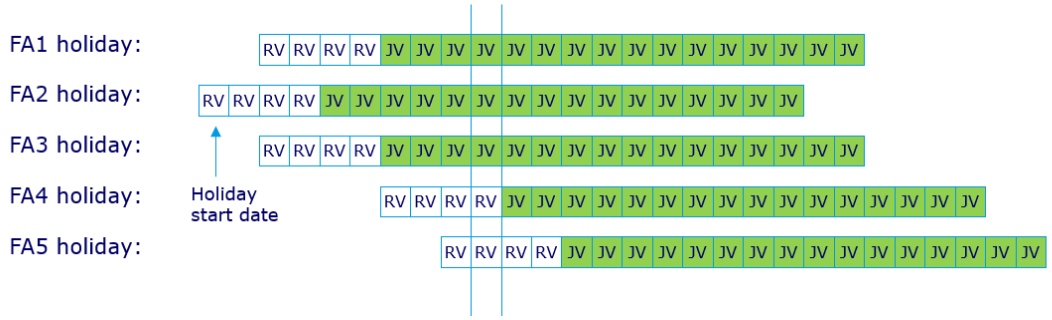

Figure 3.1: Explanation of holiday duration = RV + JV

3.2 Conflict constraints

In the formulation of the problem, there are 3 different types of conflict constraints: standard-any (std-any), standard-preference (std-pref) and priority (prio). Constraints (6) makes sure that a request in the standard block must be granted before a preferred one for the same date, while constraints (7) follow the same logic and indicate that a request in the standard block must be granted before assigning the same date to someone without a request in the standard block. Last, constraints (8) refer to the priority, which, as mentioned, is reversed, thus lower priority value should be granted before a higher priority value. In all of these constraints we use surplus variables su_c, αu_c and pu_c to allow some violations of the constraints (conflicts) and prevent our solver from leaving a large number of crew members unassigned. Respectively, the surplus variables are included in the objective function of our model with a negative sanction, or in other words as a penalty. This way, we help our model with a faster convergence based on the appropriate trade-off between assignment and violations.

For example, if $su_{1021} = 40$ it means that the employee '1021' did not get their holidays assigned on their standard requested dates and 40 other employees got their holidays assigned on these dates with a preferred request. Likewise, if $\alpha u_{2096} = 25$ it means that employee '2096' did not get their holidays assigned on their standard requested dates and 25 other employees took holidays on these dates without requesting them, but just because it was their standard block. Furthermore, if $pu_{938} = 10$ it means that employee '938' did not take holidays on their preferred requested dates and 10 other employees with lower priority got their preferred request approved for these dates.

3.3 Capacity violations

As it is mentioned in constraints (6) - (8), we use surplus variables to allow some violations and help our model run faster. That happens because our model becomes less restrictive and a trade-off is introduced between minimizing constraint violations and maximizing the objective value, which allows some extra freedom on the model's decision resulting in better solutions. Based on that, we use another category of surplus variables for the constraints (5) which is related to the capacity of the crew that is on holiday each day j, thus the capacity constraints are considered to be soft ones. We introduce 5 slack variables $cu_{n,j}$, each of them with lower bound = 0 and upper bound = 1. Thus, the total allowed violation of the capacity for each day j is equal to $cu_{1,j} + cu_{2,j} + cu_{3,j} + cu_{4,j} + cu_{5,j} \leq 5$. Furthermore, a penalty is attached to the objective function based on the violations. This penalty is scaling and that is the reason we use 5 different slack variables and not just 1, thus we have 5 different penalty coefficients (weights): W_{cu_1}, W_{cu_2}, W_{cu_3}, W_{cu_4}, W_{cu_5}. Since it is a scaling penalty we should consider the following when choosing values for the penalty coefficients:

$$0 \leq W_{cu_1} \leq W_{cu_2} \leq W_{cu_3} \leq W_{cu_4} \leq W_{cu_5}$$

$$iff$$

$$1 \geq cu_{1,j} \geq cu_{2,j} \geq cu_{3,j} \geq cu_{4,j} \geq cu_{5,j} \geq 0$$

In general, the capacity values are given by the planners of the company and they are not strict which is why we use surplus variables and allow some violations. As a result of this, an insightful suggestion can be given to the planners about increasing or decreasing the capacity for specific dates based on the outcome of the optimization process and the corresponding violations. These suggestions aim to help them with trying out different scenarios before realising the holiday schedule.

3.4 Parameters and big-M values

Determining the optimal values for penalty coefficients and the parameters is a major challenge. In this holiday assignment model, there is an important trade-off between the penalties and the reward for assigning an employee to a starting day of holidays. This trade-off must be appropriately set in order to make the model avoid leaving unassigned

employees. On the other hand, the goal is to reduce as much as possible the violations. It is indeed true that when a high penalty is assigned to capacity violations, it necessitates that the penalty for conflict violations must be lower (and vice versa). Otherwise, the model might not make optimal decisions when assigning a crew, potentially leading to conflicts that could have been avoided.

The parameter values (constants) can be adjusted depending on the desired level of strictness or optimization emphasis. In our analysis, we employed various optimization techniques, adhering to the default parameter values which are shown in Table 3.1 and were also used in the current Java model. All the results that are presented in this paper have been produced by using those values.

Parameter	**Default value**
W_{hit}	1000
W_1 standard block	1
W_2 standard request	5
W_3 preferred request	10
W_4 std-pref conflict	35
W_5 std-any conflict	1
W_6 priority conflict	35
W_{cu_1}	4
W_{cu_2}	8
W_{cu_3}	16
W_{cu_4}	32
W_{cu_5}	48

Table 3.1: Default values of parameters

The model's formulation with these presents a significant challenge. On the one hand, approving preference requests increases the objective function, on the other hand, it might cause some conflicts and penalties are attached to it. Similar trade-offs occur with standard requests and standard block assignments, since $W_3 \geq W_2 \geq W_1$. Additionally, we need to consider penalties for exceeding capacity and ignoring priority. The formulation raises several questions, such as, "Is it optimal to approve a preference request even if it leads

to conflicts?", "Should we approve requests even if they result in capacity violations?". The manual corrections, that planners might have to do after running the model, involve exploring different scenarios to resolve conflicts by swapping employees and/or increasing the capacity. The closer we get to the optimal solution, the more flexibility the planners have to explore various and better scenarios.

Constraints (6) - (8) represent 3 different categories of conflict constraints. Those constraints are considered to be in the category of the big-M constraints, and in our case, the big-M values are CS_c, CA_c and CP_c, respectively. For models with complex behaviour, like ours, it is crucial to select the lowest possible values for the big-M values [16] in order to improve the efficiency of the model. In our case, each constraint is related to an employee c, so the corresponding left-hand summation is multiplied by the big-M value and the right-hand summation includes all the other employees i except c. Based on the functionality of big-M constraints, the right-hand summation can have a maximum value equal to the big-M value.

For example in the standard-preference constraints (7), the left-hand summation is equal to CA_c if crew c was granted a request in the standard block, else 0. On the other hand, the right-hand summation is equal to the number of conflicting dates (preferred requested dates over standard requested dates) assigned which is $\leq CA_c$. Thus, in order to determine the optimal (lowest) value of CA_c we calculated the maximum number of conflicting dates (similarly with constraints (6) and (8)). Algorithm 1 demonstrates the implementation of the calculation for the big-M values used in the constraints (7) and likewise in (6) and (8).

On top of that, the Gurobi solver has several parameters that can affect the optimization process. All our experiments ran for 30 minutes in order to have a clear comparison and based on that we used the parameter `TimeLimit` = 1800. Another way to test the performance difference is the parameter `MIPGap` which indicates the minimal final gap of the optimal solution and its default value is $1e-4$. We decided on using the time parameter for the results of this thesis since it seemed to be a more accurate way of comparing performances and optimization techniques in our case, but some results of experiments with the MIPGap parameter are also provided in Section 4.7. As previously mentioned in Section 1.6, the experiments run in the Gurobi server where more jobs may be running at the same time, and if the CPU is higher than 80% then this might affect our optimization process and make it much slower, something that has a high impact on our result since we have set a time limit. However, having conducted several experiments for our baseline

Algorithm 1 Determine lowest big-M value for standard-preference conflict constraint

Input: standard_request, preference_request, employees
Output: CA: dictionary with big-M values for each employee c

```
initialize dictionary id_counts = {c : set() for c in employees}
for crew, day in standard_request_hits do
  for crew2, day2 in preference_request_hits do
    if day=day2 then
      add crew2 to id_counts[crew]
    end if
  end for
end for
initialize dictionary CA with employees
for crew, ids in items of id_counts do
  count ids and add them to CA_c
end for
return CA
```

model but also for each tested optimization technique [4], in the thesis we provide the best results we obtained, which are, obviously, the more realistic ones since it means that they were not affected at all by other jobs running in the cluster.

Moreover, in Section 4.5 we explain the impact of parameter tuning and of changing several parameters related to the optimization process and performance of Gurobi in order to improve the model, focusing on our goal to increase convergence's speed.

3.5 Results

The optimization techniques that are described in Chapter 4 were all tested for the SP rank, however, we provide the results of the current ILP Model for all different ranks. All the experiments have been done with the default parameter values of Table 3.1 and a time limit of 1800 seconds. Figures 3.2, 3.4, 3.6 and 3.8 illustrate the people that are on holidays in comparison with the available capacity and the corresponding violations, while Figures 3.3, 3.5, 3.7 and 3.7 show the exact violations of the capacity and their length because they cannot be clearly seen in the actual capacity plots due to the ranges of values.

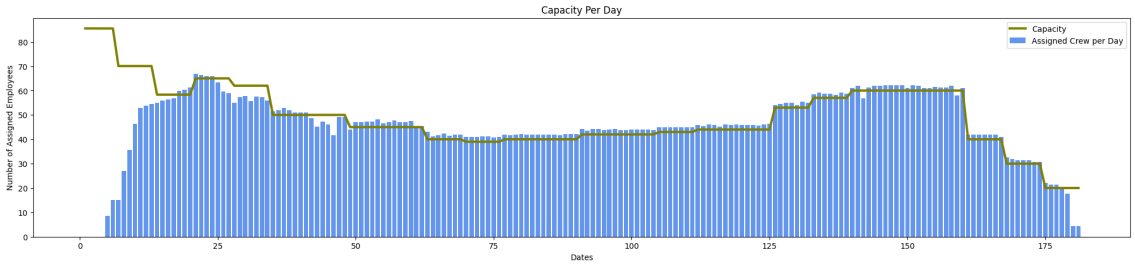

Figure 3.2: SP rank - Current ILP - capacity per day

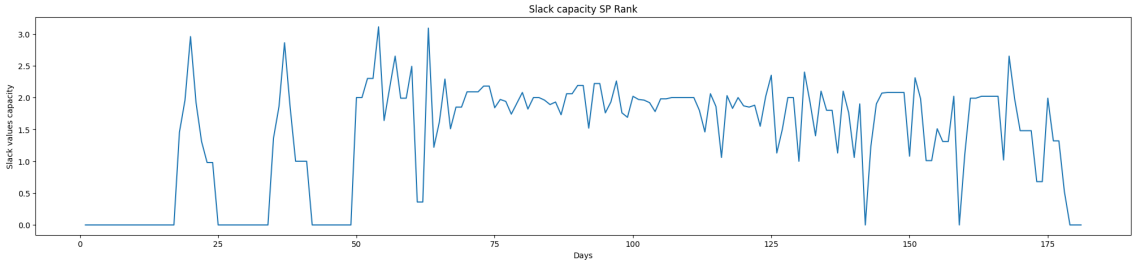

Figure 3.3: SP rank - Current ILP - capacity violations

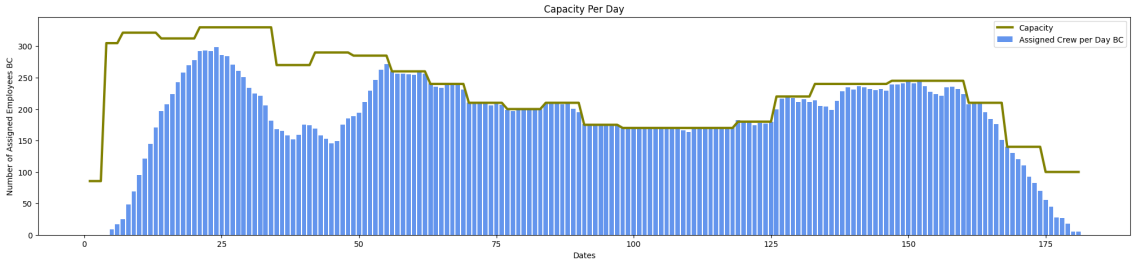

Figure 3.4: BC rank - Current ILP - capacity per day

Some extra Figures in the Appendix illustrate the conflict constraints violations. Figures 6.1, 6.4, 6.7 and 6.8 include the std-any conflicts for each rank respectively and 6.2, 6.5 and 6.9 the std-preference ones. Lastly, 6.6, 6.10, 6.3 illustrate the priority conflict constraints. Note that there are no standard-preference conflicts and priority conflicts for PU rank. The numerical results of the slack variables (= violations), as well as the number of positive values are presented in Table 3.2.

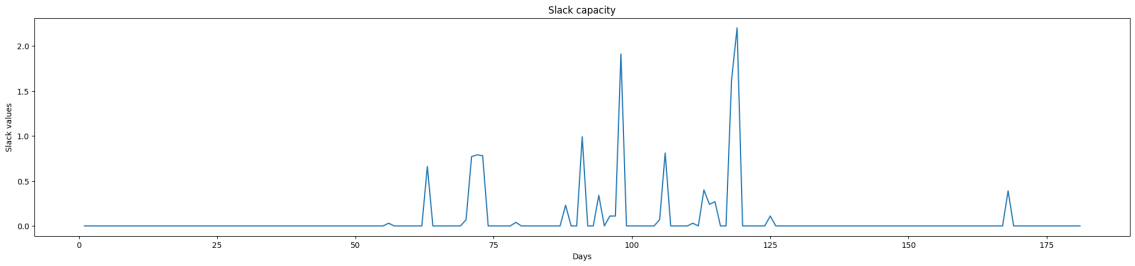

Figure 3.5: BC rank - Current ILP - capacity violations

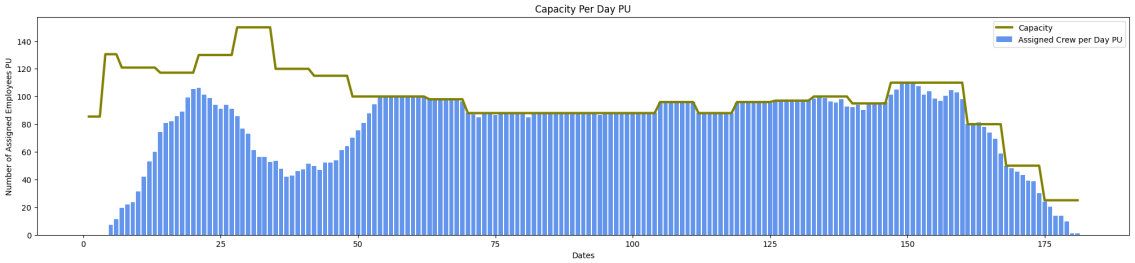

Figure 3.6: PU rank - Current ILP - capacity per day

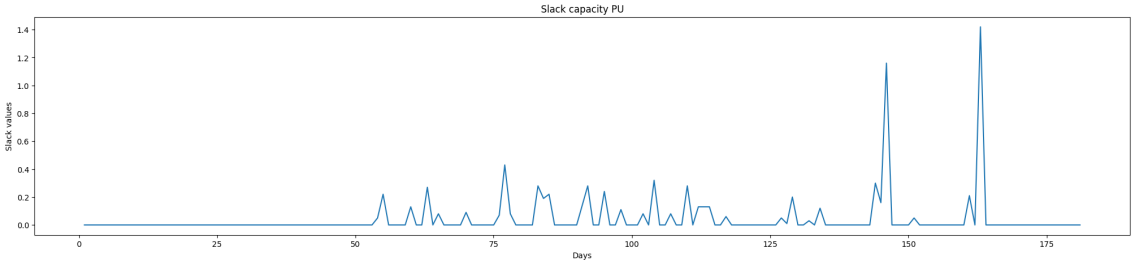

Figure 3.7: PU rank - Current ILP - capacity violations

In all the obtained figures related to the standard-any conflicts, there are many conflicts and that happens because we do not penalize that much this type of conflict. The corresponding penalty used is $W_5 = 1$ and it is obvious that Gurobi chooses to allow some violations of this conflict constraint to get closer to optimality since it does not cost that much. On the other hand, the penalty is higher for the standard-preference and priority conflicts and that is why there are either very few violations or none. However, the BC rank performs a bit differently since there are more priority violations, which is a result of the fewer capacity violations. The balance of the model is based on the trade-off between

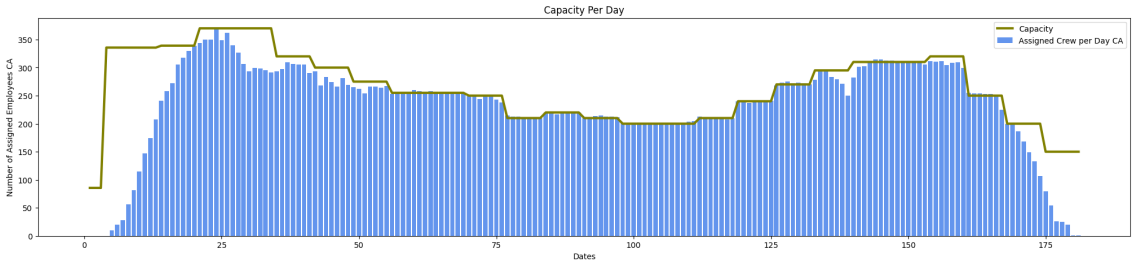

Figure 3.8: CA rank - Current ILP - capacity per day

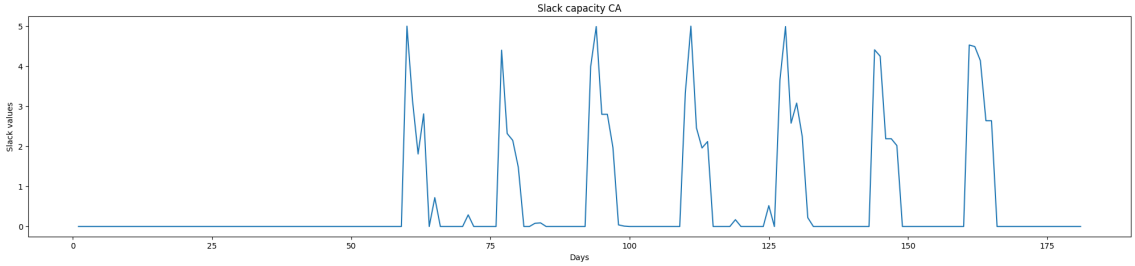

Figure 3.9: CA rank - Current ILP - capacity violations

Measurement	SP rank	BC rank	CA rank	PU rank
Total slack std-any	254.0	273.0	1716.0	66.0
Positive slack std-any variables	2	6	19	9
Total slack std-pref	4.0	19.0	46.0	0
Positive slack std-pref variables	13	2	9	0
Total slack priority	1.0	48.0	38.0	0
Positive slack priority variables	1	14	4	0

Table 3.2: Conflict constraint violations results for all ranks

the capacity and the conflict violations and for BC rank it was more profitable for the model to minimize the capacity violations by allowing some extra conflict violations.
We see in Figure 3.9 that there are some spikes and the violations reach the upper bound = 5 on some days. Considering that this plot illustrates the employees that are on holidays the corresponding days and that the average length of holidays is 17 days, we see that after approximately the 40th day it is not possible to assign any more employees on holidays. That is why the CA rank leaves some employees unassigned. Based on that, we came up

with the insightful suggestion to the planners that the given capacity for the CA rank must be increased if we want to assign all the employees, otherwise, it is not possible. We tested this assumption, by increasing the reward W_{hit} of assigning an employee to a starting day of holidays from 1000 to 10000 and the result was the same, which means that the problem occurs because of capacity issues. That is to be fixed by the planners either by increasing the capacity given (or the violations' upper bound) or by assigning each of the unassigned employees manually on top of everyone without caring for the capacity. In any way, our model is not able to fix this with the given capacity.

In Table 3.3 we present the results of all the ranks after a 30-minute run for each. Only the **CA** rank leaves some people unassigned for the capacity reasons that were mentioned before. However, each rank has some employees that are not allowed to take holidays at all (for example pregnancy reasons), so they are excluded from our data set and they are not included at all in the corresponding results. It was to be expected that **BC** and **CA** ranks would have a higher final gap because they have a larger amount of employees involved and CA is higher because some employees cannot take holidays because of capacity reasons. **SP** rank's results will be further explained and analyzed during the comparison with the optimization methods and **PU** has such a low final gap because the given capacity works well with the corresponding number of employees. In general, although the final gap of all ranks is considered to be close to optimality and the model's performance does not seem to be bad, it would be very interesting to see if and to what extent it can be further improved by applying some optimization techniques that are described in Section 4.

	SP	BC	PU	CA
Final Gap	0.0924%	0.1646%	0.0744%	0.4143%
Employees unassigned	0	0	0	30
Employees assigned	605	2897	1067	3220
Employees not allowed for holidays	0	17	5	163
Standard block approved	282	1360	405	1868
Standard request approved	275	1316	478	1124
Preferred request approved	48	221	184	228
Capacity slack	253.6514	12.96999	7.8000	100.5299
Std-any slack	254.0	273.0	66.0	1716.0
Positive slack std-any variables	13	5	9	17
Std-pref slack	4.0	19.0	0	46.0
Positive slack std-pref variables	2	2	0	9
Priority slack	1.0	48.0	0	38.0
Positive slack priority variables	1	14	0	4

Table 3.3: Results of all ranks

3.6 Evaluation

In the context of the evaluation of the model, we used the linear program (.lp files) of the model which included the objective function, the constraints and the variables used. The results of each experiment were exported as an Excel file with all the necessary information. The file contained each crew member, their block, requested dates, preferred requested dates, assignment (boolean value), standard block assignment (boolean value), standard assigned (boolean value), preferred assigned (boolean value), starting day of holidays, end date, standard-any conflicts, standard-preference conflicts, priority conflicts and a priority value. This gave us a better overview of the results in order to understand whether or not everything works correctly.

We took one step further in the evaluation and we created a verification algorithm for the evaluation of the conflict constraints. These constraints are computationally complex and our code helped us check if the conflicts were calculated in the correct way. In Algorithm 2

we evaluate the conflict constraint standard-preference. We count the actual conflicts after the optimization process for each of the crew that has some and if they are equal to the corresponding value of the slack variable that we obtain from the model, then the conflict constraint is formulated correctly and works properly. The same algorithm but with different conditions has been used for the standard-any and priority conflict constraint. Note that sometimes in the evaluation process, we face the problem of floating-point precision and sometimes the algorithm returns False because the comparison happens between two numbers very close to each other but not equal. For example, a comparison between the values 1 and 0.999999999999 returns False although the evaluation seems to be correct but this is a result of the floating-point precision. That is why we round the values to 7 decimal places.

Algorithm 2 Evaluate conflict constraints standard-preference

```
count = 0
for crew in employees do
  if slack[crew] != 0 then
    add crew's not approved standard requested dates to not_assigned
    for crew2 in employees do
      if crew2's preferred request approved for any date in not_assigned then
        count++
      end if
    end for
  end if
  if slack[crew] == count then
    print Crew has correct std-pref conflicts
  else
    print Crew does not have correct std-pref conflicts
    return False
  end if
end for
return True
```

4

Model Improvement

In this chapter we provide information about all the different optimization techniques that were used in order to improve the performance of the model. In the representation of the results, "B" stands for the Baseline model (SP rank) that was described in Section 3.

4.1 Lazy Constraints

4.1.1 Definition

Lazy Constraints represent a valuable technique used in several large-scale optimization problems, aiming to reduce complexity and improve running time. The idea is that constraints that are expected to have a low probability of impacting the solution (e.g. due to some type of redundancy) are removed from the problem before solving it. If the solution happens to violate any of these lazy constraints, the corresponding constraint is dynamically incorporated into the model and it is solved again to find the correct solution [23]. In essence, the constraints are first dormant until a feasible solution is found and become active only when it is necessary. Particularly in large-scale problems, certain constraints may not yield value during the first optimization iterations, often contributing only to increasing model's complexity.

In our case we confront a substantial volume of constraints, thus the usage of lazy constraints could be useful in our optimization process. Our proposal involves designating the conflict constraints as lazy ones. There are 3 conflict constraints per employee: standard-any conflict, standard-preference conflict and priority conflict. Thus, in total for the SP rank, where everything is tested, there are around 1700 conflict constraints. Some of these

constraints may not prove essential during the first iterations, as the initial feasible solutions might not violate them.

4.1.2 Usage and Outcome

Gurobi provides excellent functions and attributes in order to use Lazy Constraints and more specifically there are two possible ways to apply it:

- **By creating a callback function.** The callback function is being triggered during the optimization process whenever the corresponding lazy constraint is violated and the new constraint is added to the model by calling $cbLazy()$. This way, solutions that would otherwise be considered feasible can be cut off by the user. In order to apply the technique, the corresponding parameter `LazyConstraint` must be turned on by setting it equal to 1.
- **By setting the Lazy attribute.** Each constraint in Gurobi has an attribute `Lazy` which by default is equal to 0 as the constraints are treated as normal ones. This value can be changed to 1, 2 or 3 in order to define it as a lazy constraint. According to the Gurobi documentation [10] a value of 1 means that the constraint will be used to cut off a feasible solution, but it's not compulsory to be pulled in if another lazy constraint is violated. With a value of 2, in contrast, all the violated constraints will be added to the model. Last, the value of 3 means that even the constraints that cut off the relaxation solution will be added.

At first we attempted to create a callback function, in which the definition of the constraint to be subsequently added to the model must be included. Unfortunately, due to the complexity of the model, the constraint and its inequality were not linear during the optimization process and as a result, the constraints were never integrated into the model, causing the final solution to consistently violate them. The comparison under the `if statement` was not linear because of the binary variables (decision variables and indicators for the type of request) involved, as well as the big-M constraints. To address this challenge, an effort was made in order to linearize [3] both the left-hand and right-hand summations of the constraint, but in the end it was not possible, probably because of the interaction between the decision variables and the overlap in the capacity of employees

that are on holidays each day. It is essential to note again that the main decision variable of the model is $x_{i,j}$ which is equal to 1 if FA i starts holidays on day j, otherwise 0. Thus, the calculations for the capacity constraints (5) are considered complex.

The Lazy constraints technique was promising, thus we also experimented with modifying the Lazy attribute of each constraint. This approach proved successful, resulting in the addition of lazy constraints to the model, as confirmed by the linear program we generated.

The capacity violations are illustrated in Figure 4.1, 4.2, 4.3 using conflict constraints as Lazy with the corresponding value as an attribute. From these graphical representations, it is very hard to conclude which case exhibits fewer violations, but in Table 4.1 we provide all the numerical results. As baseline is defined the model, with normal constraints, that was described in Chapter 3.

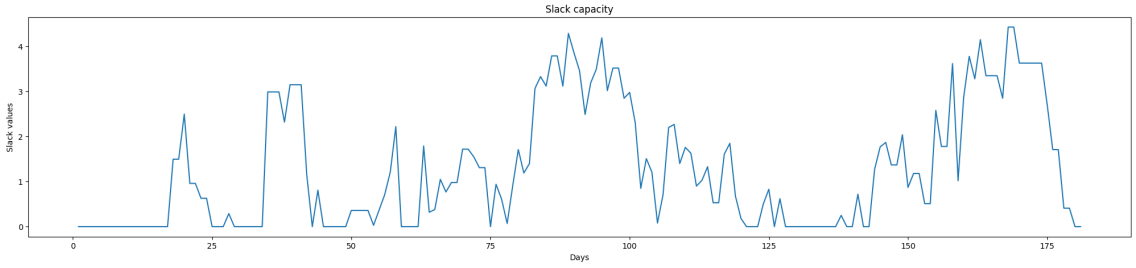

Figure 4.1: SP rank: capacity violations for conflict constraints with Lazy attribute = 1

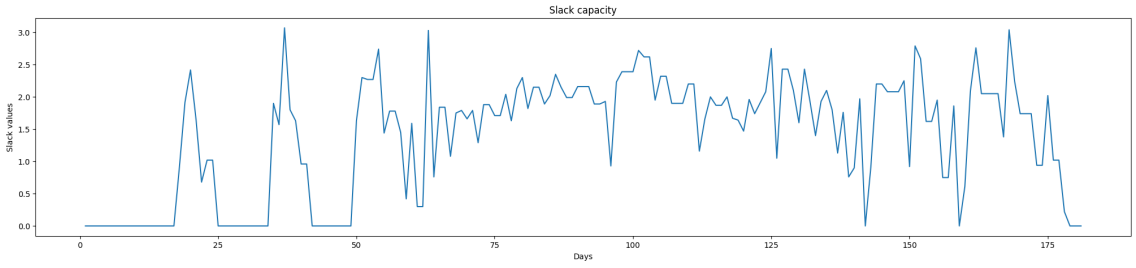

Figure 4.2: SP rank: capacity violations for conflict constraints with Lazy attribute = 2

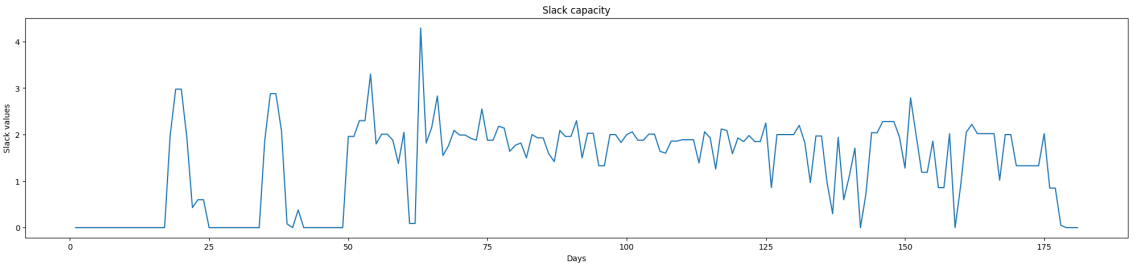

Figure 4.3: SP rank: capacity violations for conflict constraints with Lazy attribute = 3

Measurement	**B**	**All Lazy = 1**	**All Lazy = 2**	**All Lazy = 3**
Final Gap	0.0924%	2.0130%	0.0884%	0.1039%
Capacity slack	253.6514	231.09	257.4214	246.7814
Total slack std-any	254.0	449.0	290	356.0
Positive slack std-any variables	13	32	15	20
Total slack std-pref	4.0	200.0	2.0	2.0
Positive slack std-pref variables	2	32	2	1
Total slack priority	1.0	101.0	0	0
Positive slack priority variables	1	27	0	0

Table 4.1: Results and Comparison of Lazy Conflict Constraints

While it is evident that the slack capacity values remain consistent across all cases, with a slight increase in the case of Lazy = 2, the attribute Lazy = 2 outperforms for a bit the baseline model in terms of Final Gap in a run of 30 minutes. This improvement can be primarily attributed to its lower total std-any slack. In contrast, the worst model by far is the one with Lazy = 1, reaching a Final Gap of 2.013% due to exceptionally high slack values in all conflict constraints. The model with Lazy = 3 performs similarly to Baseline.

In general, the progress of the Gurobi solver in time is not deterministic because it depends on available resources. In our case, that a cluster was used, these resources are influenced by other jobs being run at the same time on the server. However, that is why the provided results are the best ones out of several experiments that have been conducted in order to

present the case that was not affected by the load of work in the cluster. On the other hand, it is true that the quality of the optimization process when we use lazy constraints is based on the first solutions that are explored because then it is checked if the constraints are violated in order to be pulled in. That might introduce some kind of uncertainty in the final gap every time we conduct a 30-minute experiment. If Gurobi explores in the early iterations some nodes (solutions) that violate the lazy constraints, it can lead to the early introduction of these constraints into the model, potentially affecting the optimization process. However, that is why the provided results are the best ones out of several experiments that have been conducted in order to present the case that was not affected by the load of work in the cluster.

4.1.3 Decomposition of big-M Constraints and Laziness

Given our observation that the conflict constraints were violated from the very first iterations, we tried an alternative approach for applying Lazy Constraints. Big-M constraints are computationally expensive for optimization techniques because their linear relaxation does not guide the solution in an optimal way. Thus, we tried to decompose the big-M constraints into individual ones in order to avoid that kind of problem and at the same time to set them as Lazy Constraints. Our idea was based on the assumption and hope that the decomposed constraints would not be violated in the very first iterations, so the technique could have an actual impact on the optimization process.

Let's say that we are looking for the standard-preference conflicts regarding employee X, who has possible conflicts with 10 more employees: Y_k with $k \in [1, 10]$. The decomposition of the corresponding big-M constraint is illustrated below.

$$CS_X \sum_j JS_{X,j} d_{X,j} - \sum_{i,j} KS_{X,i,j} d_{i,j} + su_X \geq 0$$

$$\downarrow$$

$$\sum_j JS_{X,j} d_{X,j} - \sum_j KS_{X,Y_1,j} d_{Y_1,j} + su_{XY_1} \geq 0$$

$$+$$

$$\sum_j JS_{X,j} d_{X,j} - \sum_j KS_{X,Y_2,j} d_{Y_2,j} + su_{XY_2} \geq 0$$

$$+$$

$$.$$

$$.$$

$$.$$

$$+$$

$$\sum_j JS_{X,j} d_{X,j} - \sum_j KS_{X,Y_{10},j} d_{Y_{10},j} + su_{XY_{10}} \geq 0$$

After decomposing the 1700 conflict constraints, 92490 individual constraints were created instead, something that made the building process of the problem very slow, even for the SP rank that had the least employees. More specifically, with big-M constraints the model takes around 90 seconds to be created, while with the decomposed constraints it takes around 50 minutes. Furthermore, to provide a comparison, the model creation time increased drastically from approximately 90 seconds with big-M constraints to around 50 minutes with the decomposed constraints. Moreover, the performance of the model suffered post-decomposition. Even though we define them as lazy constraints, it seems like the process of checking for possible violations in so many inequalities has a very negative impact on our model in terms of convergence. The same technique was tried with the assumption that we do not allow any violations of the conflict constraints, so without incorporating surplus variables, but the performance was still worse. Thus, it became evident that decomposing big-M constraints and categorizing them as lazy constraints did not constitute an optimal strategy.

4.2 Column Generation

4.2.1 Definition

Column generation is a similar technique to Lazy Constraints, but it focuses on the variables instead of the constraints. In simple terms, the problem is solved by iteratively adding new variables to it [21], which means that the initial problem does not contain all of them. In general, it is a useful technique when dealing with problems that include a lot

of variables and some of them might turn out to be unnecessary. The essence of the technique is about defining the initial problem and its decision variables, because sometimes it has to be reformulated. We show the entire process depicted in a flow diagram in Figure 4.4.

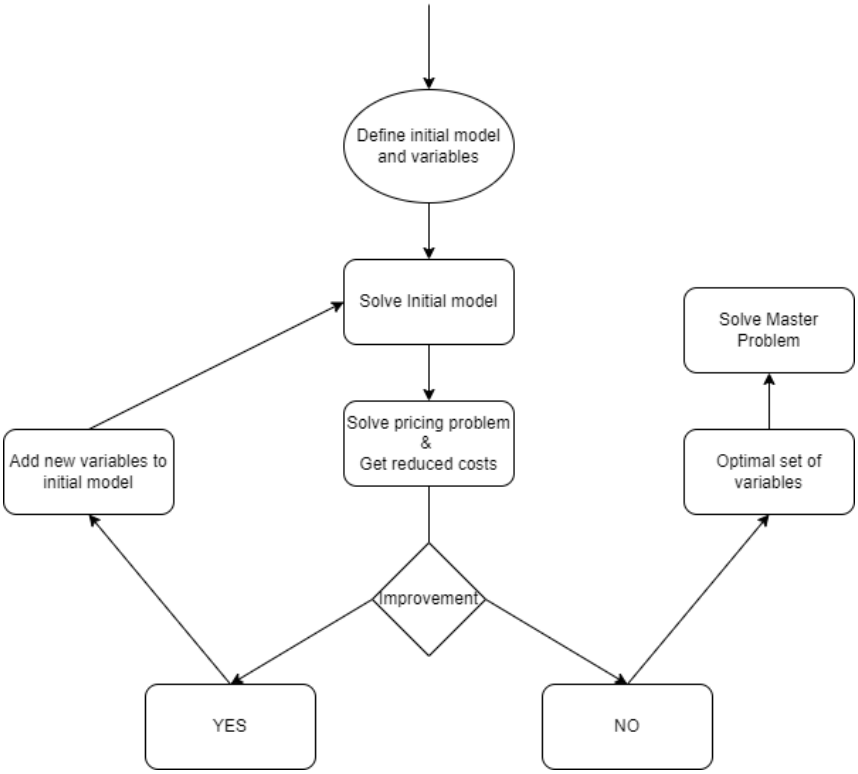

Figure 4.4: Column generation definition

As it is shown, the new variables are found by solving the pricing problem which helps us identify which of them are the most promising ones, essentially the ones that will have the greatest impact on our model's objective function. As soon as these variables are found, they are added to the model and it is solved again. This process repeats until no promising variables are found, indicating that we have reached the optimal solution for the initial model and we have the most promising set of variables. Consequently, the initial model expands with additional variables in each iteration.

It is essential to note that the initial model is not our actual master problem. More specifically, the master problem is decomposed into the initial problem and the pricing problem in order to decrease the number of variables and find the most appropriate set of them. Once this is achieved, the master problem (as it is defined in Chapter 3) is addressed

by using only the variables that have been found by the process of column generation. On top of that, when it comes to the pricing problem's functionality, it relies on dual variables [12] derived from the initial problem. When the dual variable is positive (or negative) it indicates that an increase of one unit of the right hand of the constraint will lead to this increase (or decrease) in the objective function.

4.2.2 Outcome

Although column generation is one of the most promising techniques that can be used in large-scale optimization problems, in our case it was not possible to be applied. First of all, we had to reformulate the problem because its decision variables were binary ones and in that case, it is not effective to look at the duality. We thought of many ways to reformulate it by creating continuous decision variables that encompass all potential vacation schedules, with their values indicating how frequently these schedules are utilized. Furthermore, in that case, we had to keep track of which employee took this holiday schedule so we could make sure that all the corresponding constraints were satisfied.

However, the actual issue that column generation was not possible to apply is the interaction of the variables, which in the corresponding technique are called `columns`. There is a mutual dependency between the columns, caused by the capacity constraint and conflict constraints. More specifically, in order to measure the capacity each day and satisfy the corresponding constraints, we make some calculations based on the starting date of vacation for each employee and that results in an overlap between the days. On top of that, it also causes several problems in the formulation of the other constraints since they also interact with each other. Even if we reformulate the problem and change the decision variables so they represent if an employee is on holiday on a corresponding date (and not just the starting date), the dependency will still exist. Further discussion on reformulating the problem to apply column generation will be reserved for future work in this project.

4.3 Warm Start

4.3.1 Definition

The Warm Start solution technique stands as a potential tool for enhancing our model's performance. In complex problems, providing an initial solution might make the model converge faster, particularly if it surpasses the quality of the first feasible solution found by the solver. Gurobi first analyzes the problem and, using heuristics, finds an initial feasible solution in order to start the optimization process. But what would happen if we provided another initial solution to it? The challenge of this technique lies in defining a simplified version of the corresponding complex model and solving it to optimality in order to provide its solution as the initial values to the master problem, which includes all the constraints, variables and correct parameters. However, Gurobi is a powerful solver and its initial (heuristics) solution is typically highly effective. Considering that our baseline model reaches a final gap of around 0.09%-0.10% after 30-minute experiments, something that corresponds already to a very good performance, we aimed to optimize it even a little bit by using Warm Start. It is important to acknowledge that this technique may not yield significant improvements.

Furthermore, Warm Start gives a lot of freedom and creativity so we can try out several simplified versions of the model. Simplified versions allowed us to encompass scenarios where not all constraints are included or the objective function does not take into account all the types of requests. However, it is essential to bear in mind that the simplified version of the model should be capable of reaching optimality or a predefined optimal point very quickly so the master problem is solved afterwards within the 30-minute experiment time frame.

4.3.2 Simplified version with specific requests

We tried five different simplified versions of our model in order to give a warm start solution to our master problem. First, we present the performance of our chosen simplified versions:

1) Simplified version: Include only preference requests.

Here, our simplified model included only the preference requests in the objective function.

$$\begin{aligned}\max \quad & (W_{hit} + W_3) \sum_{i \in C, j \in PR_i} d_{i,j} - W_{cu_1} \sum_{j \in D} cu_{1,j} - W_{cu_2} \sum_{j \in D} cu_{2,j} \\ & - W_{cu_3} \sum_{j \in D} cu_{3,j} - W_{cu_4} \sum_{j \in D} cu_{4,j} - W_{cu_5} \sum_{j \in D} cu_{5,j}\end{aligned}$$

But this simplified version seemed to be very slow, since after 10 minutes it reached a Final Gap of 0.49%. Taking into account that the simplified version must reach optimality very fast so Gurobi can start solving the master problem, this case did not perform well. That happened because there are a lot of preference requests for specific days (the most popular ones) and we allow only preferred assignments, thus capacity issues arise and Gurobi struggles to find the optimal solution within a reasonable time.

2) Simplified version: Include only standard and preference requests.

Extending our first try, here we included the standard requests as well in order to give Gurobi more flexibility to reach optimality.

$$\begin{aligned}\max \quad & (W_{hit} + W_2) \sum_{i \in C, j \in SR_i} d_{i,j} + (W_{hit} + W_3) \sum_{i \in C, j \in SR_i} d_{i,j} - W_{cu_1} \sum_{j \in D} cu_{1,j} \\ & - W_{cu_2} \sum_{j \in D} cu_{2,j} - W_{cu_3} \sum_{j \in D} cu_{3,j} - W_{cu_4} \sum_{j \in D} cu_{4,j} - W_{cu_5} \sum_{j \in D} cu_{5,j}\end{aligned}$$

The simplified version reached optimality within 11 minutes and we managed to import its solution as a warm start to the master problem.

3) Simplified version: Include only standard block assignments.

We decided to check out what happens if we include the standard block assignments as the only possible ones:

$$\begin{aligned}\max \quad & (W_{hit} + W_1) \sum_{i \in C, j \in V_i} d_{i,j} - W_{cu_1} \sum_{j \in D} cu_{1,j} - W_{cu_2} \sum_{j \in D} cu_{2,j} \\ & - W_{cu_3} \sum_{j \in D} cu_{3,j} - W_{cu_4} \sum_{j \in D} cu_{4,j} - W_{cu_5} \sum_{j \in D} cu_{5,j}\end{aligned}$$

Here, it managed to produce the optimal solution within 5 minutes, thus we could proceed to the master problem.

4) Simplified version: Include only standard block and standard requests. Another combination of requests is the standard block and the standard requests.

$$\begin{aligned}\max \quad & (W_{hit}+W_1)\sum_{i\in C, j\in V_i} d_{i,j} + (W_{hit}+W_2)\sum_{i\in C, j\in SR_i} d_{i,j} - W_{cu_1}\sum_{j\in D} cu_{1,j} \\ & - W_{cu_2}\sum_{j\in D} cu_{2,j} - W_{cu_3}\sum_{j\in D} cu_{3,j} - W_{cu_4}\sum_{j\in D} cu_{4,j} - W_{cu_5}\sum_{j\in D} cu_{5,j}\end{aligned}$$

The objective function of this simplified version seemed to be slow, since after 30 minutes it reached a final gap of 0.0267%. However, since it got very close to optimality, we decided to run it again and set optimal gap = 0.03%. Thus, after 9 minutes it reached our set optimality.

5) Simplified version: Include only standard requests Our last try to define a simplified version was the one that includes only the standard requests.

$$\begin{aligned}\max \quad & (W_{hit}+W_2)\sum_{i\in C, j\in PR_i} d_{i,j} - W_{cu_1}\sum_{j\in D} cu_{1,j} - W_{cu_2}\sum_{j\in D} cu_{2,j} \\ & - W_{cu_3}\sum_{j\in D} cu_{3,j} - W_{cu_4}\sum_{j\in D} cu_{4,j} - W_{cu_5}\sum_{j\in D} cu_{5,j}\end{aligned}$$

This case seemed to perform well and reached optimality within 1 minute and we were able to continue to the master problem.

4.3.3 Results of Master Problem

Only our first simplified version did not manage to reach optimality, thus we were able to use four different warm start solutions and check the performance of the master problem to figure out if it got closer to the optimal solution.

The detailed results are shown in Table 4.2 and we can notice that there is no significant improvement. All the warm start solutions did not make an impact on the master problem's performance. Only the 4th case, where in the simplified version we considered only the standard block assignments and standard requests, performed almost the same as the Baseline model, but we do not consider this as a significant improvement. In Figure

4.5, 4.6, 4.7 and 4.8 we present the corresponding capacity violations of each and we can see that there is not that much difference between the cases, only in the 4th one there are a bit fewer spikes and that was to be expected since the total capacity slack is a bit lower.

Measurement	B	1	2	3	4	5
Final Gap	0.0924%	(-)	0.1338%	0.1192%	0.0905%	0.1178%
Std block approved	282	(-)	275	274	295	281
Std request approved	275	(-)	280	279	274	275
Pref request approved	48	(-)	50	52	36	49
Capacity slack	253.6514	(-)	254.4814	260.9214	247.3814	255.0014
Total slack std-any	254.0	(-)	248.0	270.0	289.0	286.0
Positive slack std-any variables	13	(-)	14	14	14	15
Total slack std-pref	4.0	(-)	5.0	5.0	2.0	2.0
Positive slack std-pref variables	2	(-)	3	3	1	1
Total slack priority	1.0	(-)	2.0	0	0	2.0
Positive slack priority variables	1	(-)	1	0	0	1

Table 4.2: Warm Start Master problem results

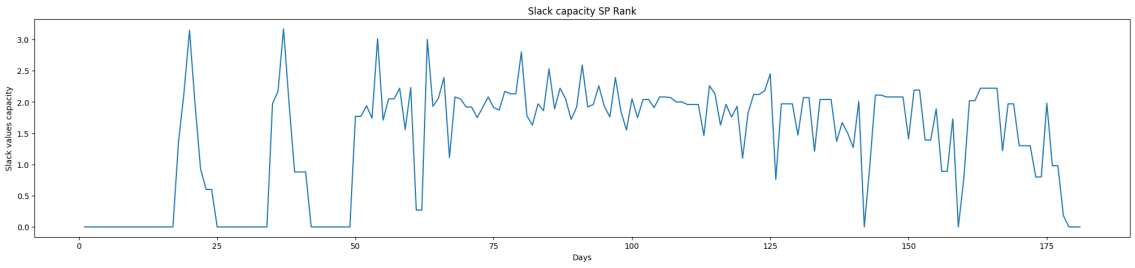

Figure 4.5: Capacity violations - Warm Start: standard & preference requests (case 2)

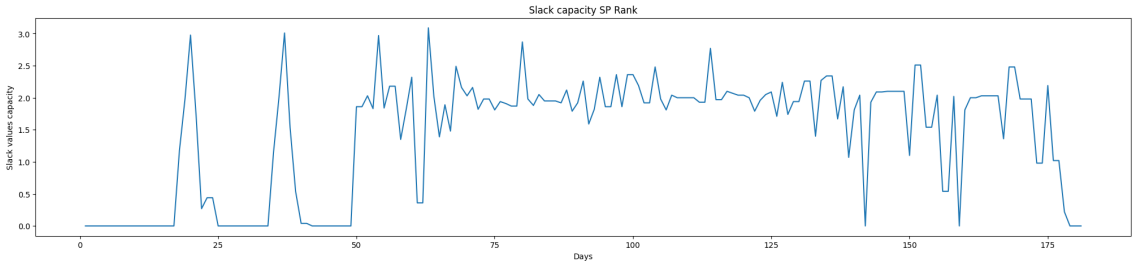

Figure 4.6: Capacity violations - Warm Start: standard block only (case 3)

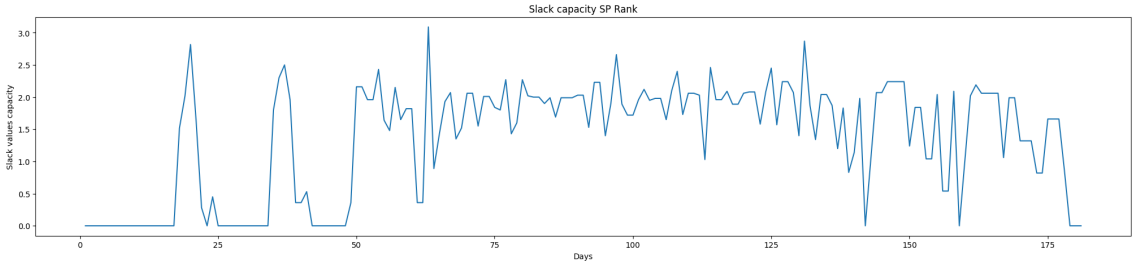

Figure 4.7: Capacity violations - Warm Start: standard block & standard requests (case 4)

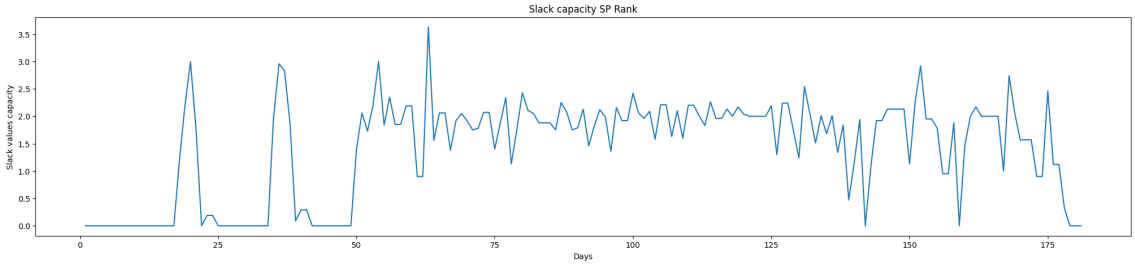

Figure 4.8: Capacity violations - Warm Start: standard requests only (case 5)

4.4 Multiple Objective Functions

4.4.1 Definition

It is clear that the problem and its objective function encompass various terms, including different types of requests but also the corresponding penalties, while plenty of constraints associated with each of these terms. Thus, we are able to decompose the objective functions into multiple components, effectively transforming the problem into a Multi-objective optimization problem. Multi-objective optimization requires also the corresponding weights

of each function used. In our case, we have weights for the requests and penalties for the violations, but instead of specifying them as weights of each function, we can just include them in the sub-functions. It is also essential to establish the priority of each objective function, or in other words which is the order that the objective functions are going to be optimized. Below there is a simple example of how a single-objective optimization problem can be transformed into a Multi-objective one.

$$\max(f_{(x)} + g_{(x)} - z_{(x)}) = \begin{cases} \max(f_{(x)}) + \max(g_{(x)}) - \max(z_{(x)}) \\ \textbf{OR} \\ \max(f_{(x)}) + \max(g_{(x)}) + \min(z_{(x)}) \end{cases}$$

4.4.2 Objective Functions Sets and Outcome

The technique proves highly valuable when dealing with multiple decision-making factors, such as how many standard requests should be approved while maximizing the preferred requests and minimizing the penalties. In our case, the complexity arises from the interconnected nature of these request types, stemming from capacity limitations and conflict constraints. That is why we tried many combinations of objective functions, from very simple to very complex ones but keeping a limitation of having a maximum of 3 objective functions at a time. We operated on the assumption that having more than enough objective functions might make Gurobi search for unnecessary nodes to reach optimality in the first sub-objective functions that will probably not be needed for the next ones. Our goal is to optimize the problem in terms of computational time and performance and this necessitates maintaining a formulation that is as streamlined as possible. Below, we outline the different combinations that we tried and the corresponding results.

1)Preferred and Standard requests, 2)Everything else. First, we tried to maximize the preferred requests since these are the ones that have higher weight (1000 for hit + 10 for preferred request) in combination with the standard requests because they are very important for the left-hand summations in the conflict constraints. The first objective function reached optimality within 12 seconds, something that was expected since we just maximized the preferred and standard requests without taking into account standard blocks and penalties. Then, on the other hand, the second objective function ran for 30 minutes and reached a Final gap of 13.2376 % which is very high.

2)Standard requests and Standard block, 2)Everything else. This case resulted in a final gap of 0.17% after a 30-minute run for the first objective function, thus the second one did not even start since the first did not reach optimality.

3)Standard requests, 2)Everything else. Similar to our first try, here we tried to maximize first the standard requests since they are the ones that play an important role in the left-hand summation of the conflict constraints. After 30 minutes the first objective function reached a final gap of 0.0193% which is very close to optimal, but on the other hand, it seemed like the second one would be very slow.

4)Standard block, 2)Everything else. On the other hand, maximizing first the standard block assignments was a good idea since the first objective function reached optimality within 17 seconds. Afterwards, the second one reached 0.27% after 30 minutes, which is not very high but far from optimal and worse than our baseline model.

5)All requests, 2)Penalties Here we tried to maximize first all the requests, no matter the type and then all negative penalties. Note that in the second objective function, we maximize the negative penalty, so as to be as close as possible to zero. This can be done also by minimizing the penalties without using a negative sanction. The first optimization problem achieved optimality within 10 seconds but the second one did not perform well because after 30 minutes the final gap was 34.0%.

6)Preferred requests, 2)Standard requests, 3)Everything else This was our first try with 3 different objective functions. The first one reached optimality almost immediately, while the second one needed just 2 seconds. Unfortunately, the third one made it to a final gap of 1.99% after 30 minutes, thus we were too far from the optimal solution.

7)All requests, 2)Capacity penalties, 3)Conflict penalties Another try with 3 different objective functions, but it performed worse than the previous one. The first objective function needed 11 seconds for optimality, but the second one reached 0.96% after 30 minutes and based on that the third one never started.

The results of all our cases are summarised in Table 4.3

Case	First Obj. Function	Second Obj. Function	Third Obj. Function
Case 1	Optimality within 12 seconds	13.2376 %	`Does not exist`
Case 2	0.17%	`Not reached`	`Does not exist`
Case 3	0.0193%	`Not reached`	`Does not exist`
Case 4	Optimality within 17 seconds	0.27%	`Does not exist`
Case 5	Optimality within 10 seconds	34.0 %	`Does not exist`
Case 6	Optimality immediately	Optimality within 2 seconds	1.99%
Case 7	Optimality within 11 seconds	0.96%	`Not reached`

Table 4.3: Results of Multiple Objective Functions

4.5 Parameter Tuning

Gurobi offers a range of parameters that can play an important role in the optimization process, especially complex formulations like ours. Instead of trying out random changes in some parameter values, we let Gurobi solve the problem for many different sets of parameters by conducting a systematic exploration and giving us the results compared to the baseline model (current formulation). The experiments were run again for a duration of 30 minutes, using the values that are mentioned in table 3.1 and the results are presented in table 4.5, while in table 4.4 the default values of the parameters are mentioned. For the two cases that the value was close to the final gap of the current model (=0.0923%) we repeated the model runs by setting these values explicitly in order to obtain more accurate results, because the parameter tuning process returns the final gap rounded to two decimal places.

To provide more details, based on Gurobi's documentation, here is an explanation of some specific parameters that were explored, which significantly influence the solver's behavior and the quality of the solutions it generates:

- `CutPasses`, `GomoryPasses`, `Cuts`: These parameters are related to the specific cuts that are implemented in the optimization process of the branch-and-cut technique that Gurobi uses [11].
- `AggFill`: controls the amount of fill allowed during presolve aggregation.
- `Aggregate` controls exactly the aggregation level in presolve.

Parameter	Default Gurobi Value
CutPasses	-1
AggFill	-1
Aggregate	1
GomoryPasses	-1
MIPFocus	0
PrePasses	-1
Symmetry	-1
VarBranch	-1
Heuristics	0.05
Cuts	-1
PreSparsify	-1
Presolve	-1
NormAdjust	-1

Table 4.4: Default values of parameters in Gurobi

- `PrePasses`: limits the number of passes performed.
- `MIPFocus`: different values of this parameter make Gurobi focuses on feasibility or optimality.
- `Heuristics`: establishes the time spent on MIP heuristics.
- `Presolve`: controls the presolve level.
- `Symmetry`: controls the symmetry detection.
- `VarBranch` controls the selection strategy of the branch variable
- `PreSparsify`: controls the presolve sparsify reduction
- `NormAdjust`: plays a role in selecting among multiple pricing norm variants.

We see from the results that the most promising set of parameters is the one with Heuristics 0.5 and MIPFocus 2 which has a Final Gap of 0.0895% when the baseline model has 0.0923%, thus there is some improvement but the difference is not profoundly significant. The corresponding setting of MIPFocus focuses more on proving optimality, so it might force the model to run faster than before and in combination with the `Heuristics` value the model's performance is improved a bit. With this value of `Heuristics`, we let

Parameters	Final Gap	Re-run Final Gap
1) Current Model	0.0923%	Not needed
2) CutPasses 1	0.15%	Not needed
3) AggFill 100	0.11%	Not needed
4) Aggregate 0	0.11%	Not needed
5) GomoryPasses 0	0.18%	Not needed
6) MIPFocus 2	0.11%	Not needed
7) PrePasses 1	0.13%	Not needed
8) Symmetry 2	0.17%	Not needed
9) MIPFocus 2, VarBranch 3	0.11%	Not needed
10) Heuristics 0.5	0.16%	Not needed
11) Heuristics 0.5, MIPFocus 2	**0.9%**	**0.0895%**
12) GomoryPasses 1	0.13%	Not needed
13) Symmetry 2, GomoryPasses 0	0.18%	Not needed
14) Heuristics 0.001	0.14%	Not needed
15) Cuts 1	0.17%	Not needed
16) Heuristics 0.001, Symmetry 2	0.10%	Not needed
17) PreSparsify 1	0.14%	Not needed
18) Presolve 2	**0.09%**	**0.1091%**
19) NormAdjust 3	0.10%	Not needed

Table 4.5: Sets of Parameters tried out

more MIP runtime be devoted to heuristics so better feasible solutions are produced and later on Gurobi can focus better on optimality based on that. The case where Gurobi used Presolve2 made it to 0.1091% in the second run, slightly exceeding 0.09% but that happened because the presolve process takes more time and some kind of uncertainty is involved in that. All the results are illustrated in Figure 4.9.

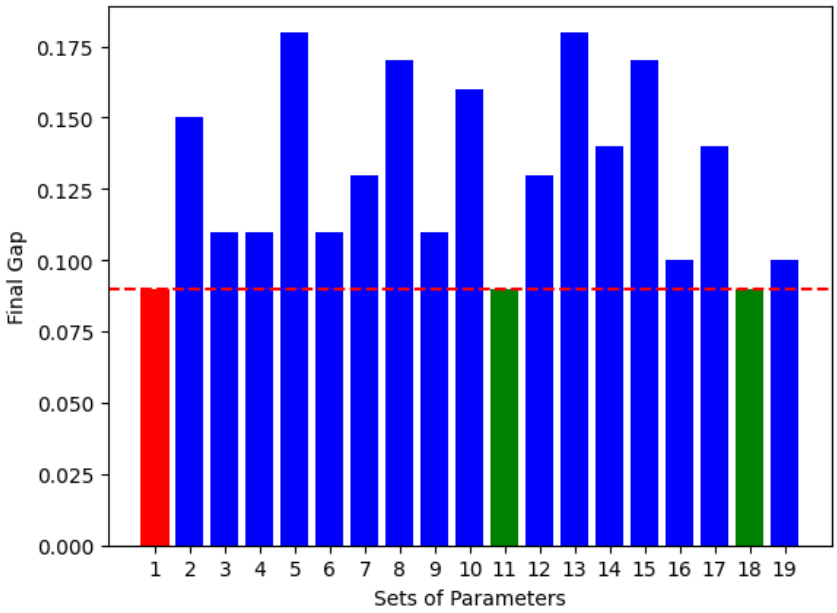

Figure 4.9: Final Gap of each case of parameter tuning

4.6 Relaxed weekly capacity

4.6.1 Definition and Reasons

Aiming to improve the model's performance, the last idea that we had was to consider the measurement of the weekly violations of the capacity. We believed that by tracking the total violations on a weekly basis, we could potentially guide the model towards the optimal solution by following another path. We experimented with three different approaches, which are elaborated further in the following subsections:

- Use of weekly capacity constraints without upper bound in the capacity slack variables.
- Addition of extra 7-days constraints alongside the daily ones. By 7-days we mean every single range of 7 days in the period.
- Combination of 7-days constraints and laziness.

These strategies aimed to improve our model's performance by considering the weekly violations of the capacity as a crucial metric.

4.6.2 Weekly capacity constraints and no daily upper bounds

The current model includes five slack variables with a lower bound of 0 and an upper bound of 1. As previously explained in Chapter 3.3, this setup introduces a scaling penalty with different coefficients in the objective function. Trying to make the model follow another path towards optimality, we decided to give some freedom to the capacity violations and remove the upper bound of the slack variables. Instead, we added constraints to limit the values of each slack variable per week while keeping the concept of scaling penalty with the different weights which are mentioned in Table 3.1. Thus, we tried the two following cases:

1. $\sum_{j \in w} cu_{1,j} \leq 7$, $\sum_{j \in w} cu_{2,j} \leq 7$, $\sum_{j \in w} cu_{3,j} \leq 7$, $\sum_{j \in w} cu_{4,j} \leq 7$, $\sum_{j \in w} cu_{5,j} \leq 7$, for each week w.

2. $\sum_{j \in w} cu_{1,j} \leq 4$, $\sum_{j \in w} cu_{2,j} \leq 4$, $\sum_{j \in w} cu_{3,j} \leq 4$, $\sum_{j \in w} cu_{4,j} \leq 4$, $\sum_{j \in w} cu_{5,j} \leq 4$, for each week w.

Here, in the context of these adjustments, where we removed the daily upper bound and introduced new constraints, it is obvious that the final gap does not matter. Our primary focus is on assessing the overall performance and basically the capacity violations, but also the conflict ones. In Table 4.6 we provide the detailed results for each case. We see that we achieved a low final gap for the first case but, as mentioned before, it does not have an impact. On the other hand, we observed a lot of conflict violations in the second case. The comparison of those two cases with our baseline model is illustrated in Figure 4.10. We notice that both cases with the weekly upper bound instead of the daily one result in many spikes. It is indeed true that on many days the number of violations is below the number of the corresponding ones for the baseline model, but we also observe some very high values. This outcome is less than ideal, because it does not offer practical guidance and cannot be translated into an insightful suggestion to the planners regarding an increase in the capacity due to the not equal spread of the violations. Planners cannot increase the capacity by 6 units for example on specific days. Instead, capacity adjustments typically occur in a continuous, weekly range. In conclusion, this approach does not provide actionable insights for managing capacity effectively due to the uneven distribution of violations.

Measurement	weekly upper bound = 7	weekly upper bound = 4
Final Gap	0.0432%	0.2164%
Std block approved	279	291
Std request approved	280	270
Pref request approved	46	44
Capacity slack	249.9414	223.8242
Total slack std-any	285.0	389.0
Positive slack std-any variables	13	17
Total slack std-pref	2.0	4.0
Positive slack std-pref variables	1	2
Total priority variables	0	4.0
Positive priority variables	0	1

Table 4.6: Weekly upper bound instead of daily ones

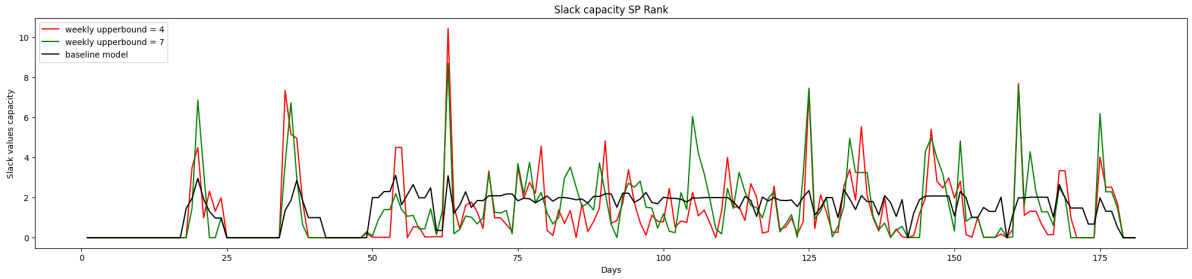

Figure 4.10: Capacity violations for weekly upper bounds

4.6.3 7-days and daily capacity constraints

Our second approach was to keep the daily capacity constraints but add next to them some limitations for the total violations in a range of days. Thus, the capacity constraints that are described in Chapter 3 remained on usage and the maximum allowed daily violations were equal to $cu_{1,j}+cu_{2,j}+cu_{3,j}+cu_{4,j}+cu_{5,j} \leq 5$. However, having seen in all our previous experiments that most of the time there are 2 violations per day on average, we aimed to narrow down the feasible region. Instead of lowering the upper bounds of the capacity slack variables, we introduced a new set of constraints for the total violations every 7 days. It is essential to clarify that these constraints are not associated with weeks, but with every single range of 7 days that lies inside our period. Thus, there was a constraint for the violations between the days 1-7, another one between the days 2-8 and so on. The summer

period includes 181 days, which is why we had 175 7-days constraints. An illustration of the 7-days constraints approach is given in Figure 4.11.

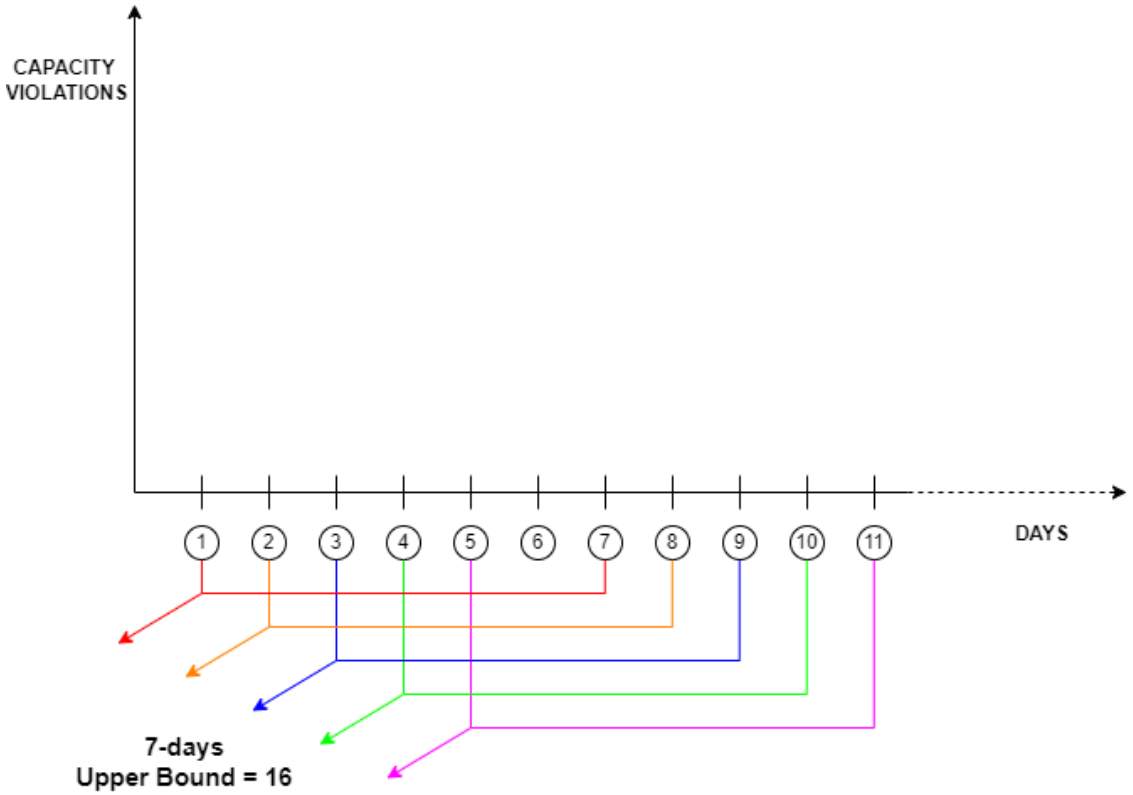

Figure 4.11: Example of 7-days upper bound = 16

Based on the baseline model that includes only the daily capacity constraints, we can calculate that the maximum allowed capacity violations every 7 days is equal to 35. Furthermore, based on our observation that the average daily violations are equal to 2, we thought that we have to set a limitation for the 7-days violations which would be approximately half of this value. In Table 4.7 we present the results of all the cases that we tried where `ub` stands for upper bound and `B` denotes the baseline model. The results include the final gap, the number of approved requests for each type, and the slack values corresponding to the violations.

As we can see the optimal value of the 7-days upper bound is close to 16, because this is the case that outperforms the baseline. We see that this case has a lower final gap, although we see similar values in the rest of the measurements. Using $ub = 16$ we get a bit closer

Measurement	B	ub=14	ub=15	ub=16	ub=17	ub=18	ub=20
Final Gap	0.0924%	0.2058%	0.1974%	0.0809%	0.1022%	0.1053%	0.1316%
Std block approved	282	277	271	288	308	285	298
Std request approved	275	276	278	276	265	275	271
Pref request approved	48	52	56	41	32	45	36
Capacity slack	253.6514	231.1914	255.7814	251.2114	240.8014	251.1514	252.4314
Total slack std-any	254.0	292.0	277.0	254.0	328.0	301.0	310.0
Positive slack std-any variables	13	14	15	13	16	16	15
Total slack std-pref	4.0	16.0	6.0	4.0	0.0	2.0	0.0
Positive slack std-pref variables	2	6	4	2	0	1	0
Total slack priority	1.0	15.0	12.0	0.0	2.0	1.0	0.0
Positive slack priority variables	1	5	6	0	1	1	0

Table 4.7: Weekly relaxation cases

to the optimal solution because it looks like there are more days that the level of violation is below the baseline model than above, as illustrated in Figure 4.12. This means that there are lower ones per day so a lower penalty is attached to the objective function since we have a scaling penalty. In Figure 4.13 and 4.14 we see a similar case with the upper bounds ("WU") of 17 and 18, but looking at Table 4.7 we see that the total slack std-any is higher. On top of that, although the upper bound of 17 has a lower total capacity slack, it approves fewer preferred requests (they give +10 to the objective function). Thus, these two cases reach a final gap which is a bit higher than the baseline model. We conclude that the case with an upper bound of 16 produces an improved solution compared to the baseline model but there are still a lot of capacity and conflict violations.

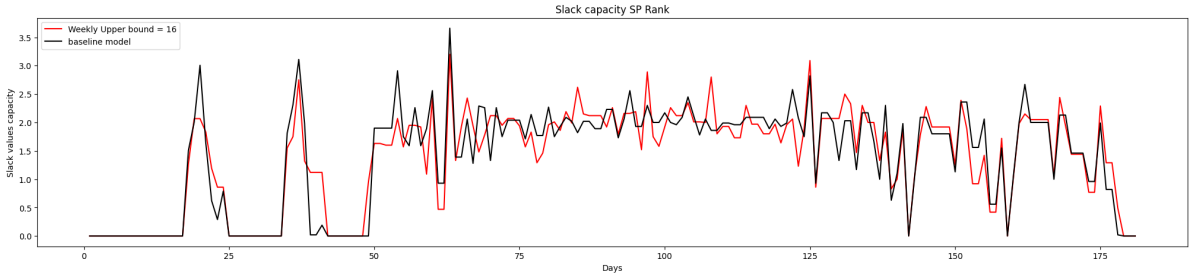

Figure 4.12: Comparison of capacity violations: Baseline & WU 16 cases

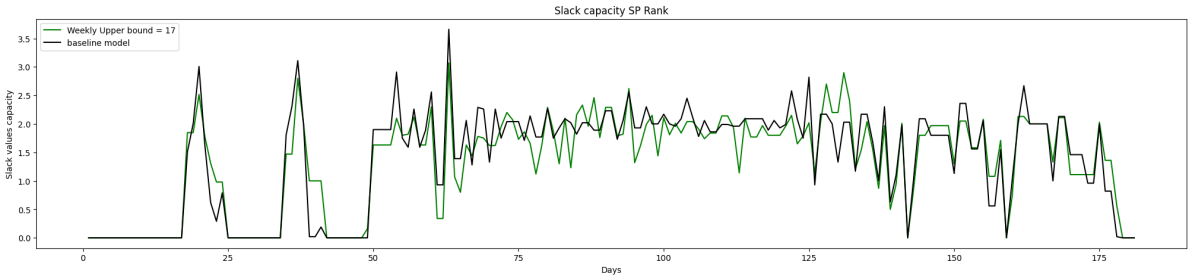

Figure 4.13: Comparison of capacity violations=: Baseline & WU 17 cases

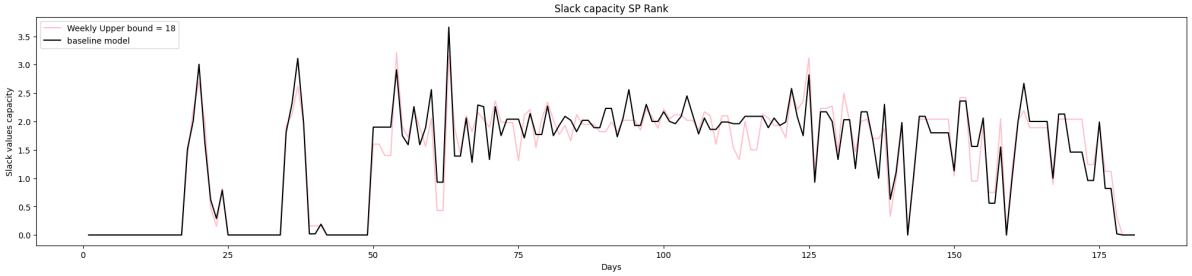

Figure 4.14: Comparison of capacity violation: Baseline & WU 18 cases

4.6.4 7-days and daily capacity lazy constraints

Extending our idea about the 7-days constraints, we understood that the daily capacity constraints might be unnecessary for some days. Thus, we tried to combine this technique with using the daily ones as Lazy Constraints. Based on that, we tested out the three most promising cases of the 7-days constraints: weekly upper bound ("WU") 15, 16 or 17. For each of those we set the daily capacity constraints' attribute `Lazy` equal to 2 or equal to 3, so in total we had 6 cases. The detailed results are presented in Table 4.8 and the capacity violations are illustrated in Figure 4.15, 4.16 and 4.17 with a comparison between the corresponding cases and the baseline model. We might have obtained some lower Final Gaps compared to the baseline model, but still, there are a lot of conflict violations which means that it will not make any difference in the effort of the planners when they try out different scenarios. Furthermore, as we can see from the visualization of the slack capacity values, all the cases perform almost the same, since some days have a bit higher violations and some others a bit lower and all of them follow almost the same pattern.

Measurement	WU = 15, Lazy = 2	WU = 15, Lazy = 3	WU = 16, Lazy = 2	WU = 16, Lazy = 3	WU = 17, Lazy = 2	WU = 17, Lazy = 3
Final Gap	0.2781%	0.0937%	0.1316%	0.0979%	0.0963%	0.0869%
Std block approved	332	300	288	288	293	281
Std request approved	248	268	272	272	273	277
Pref request approved	25	37	45	45	39	47
Capacity slack	230.2914	242.9214	254.5914	248.4514	247.7314	252.4214
Total slack std-any	538.0	333.0	306.0	304.0	292.0	247.0
Positive slack std-any variables	25	17	16	15	16	16
Total slack std-pref	0	2.0	2.0	2.0	4.0	4.0
Positive slack std-pref variables	0	1	1	1	2	2
Total slack priority	15.0	0	3.0	1.0	0	2.0
Positive slack priority variables	10	0	3	1	0	2

Table 4.8: 7-days and laziness

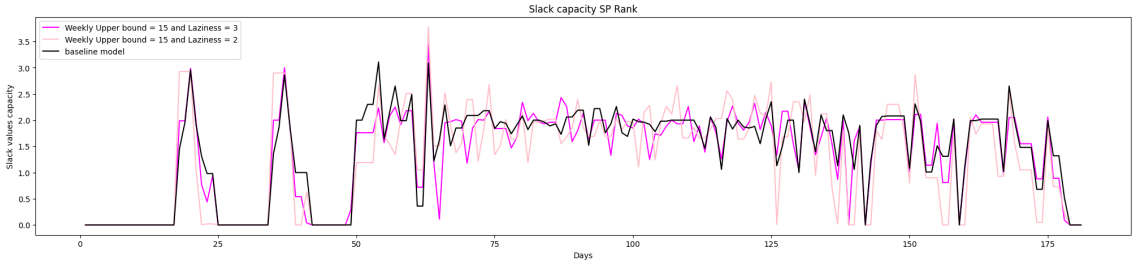

Figure 4.15: Comparison of capacity violations with Laziness: Baseline & WU 15 cases

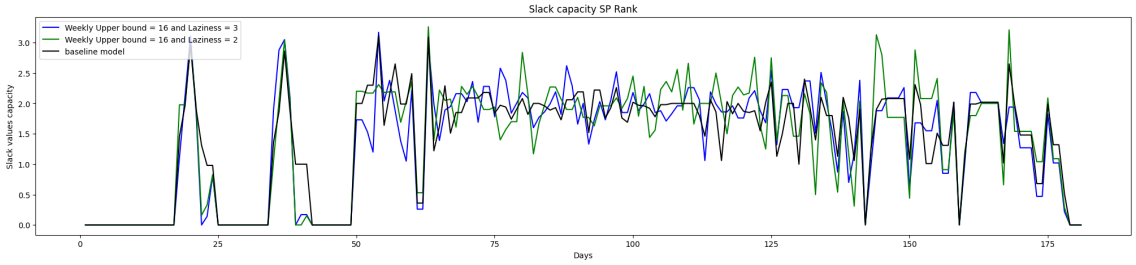

Figure 4.16: Comparison of capacity violations with Laziness: Baseline & WU 16 cases

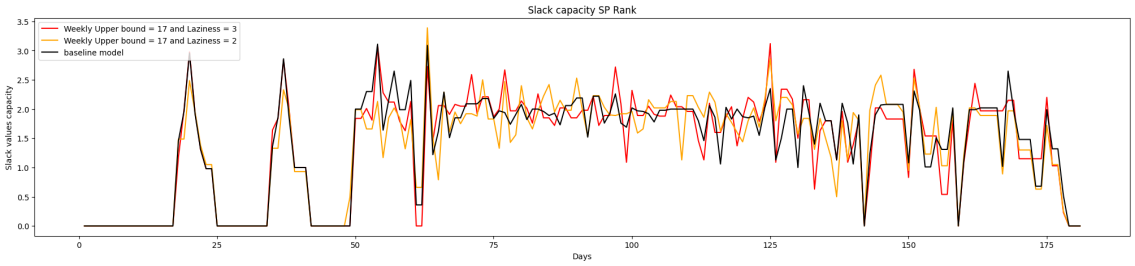

Figure 4.17: Comparison of capacity violations with Laziness: Baseline & WU 17 cases

4.7 MIPGap experiments of best cases

It is, indeed, true that using a time limit, the quality of the solution also depends on the computer or the server where the experiment is conducted. On the other hand, although we compare solutions that have been found within a certain time limit, this way we gain an understanding of how useful is the technique we applied by checking the results in detail. In this section, we will show the results of the experiments we conducted using the `MIPGap` parameter as a terminator criterion instead of the `TimeLimit` in order to eliminate the dependency between the quality of the solution and the performance of the hardware. Thus, we set an optimal final gap of 0.085% and there is no time limit of 1800 seconds now. Based on that, we decided to conduct experiments for the baseline model and the 3 most promising cases that we found in our Model Improvement applications: 1) All Lazy = 2, 2) 7-days upper bound = 16, and 3) 7-days upper bound = 17 with Lazy = 3, and to provide a comparison between them to find out whether there is an actual approach that improves the performance. This way we aim to compare the best cases found without taking into consideration the time spent for them, nor other possible jobs that might be running in the cluster and affect the performance. We want to compare the baseline with 3 other optimization techniques only in terms of the violations.

As we can see in the detailed results, that are provided in Table 4.9, all experiments gave almost the same solution in terms of quality. That was to be expected since for all of them the Final Gap was set to 0.085%, but we wanted to check what happens with the violations. It is clear that the 3 optimization techniques performed the same as the Baseline model. Only the "All Lazy = 2" case could be defined as a potential improvement since there were 16 conflict violations while the Baseline had 17 (that is why the difference in the Total slack std-any). More specifically, it has 2 fewer violations for the std-any but 1

Measurement	Baseline	All Lazy = 2	7-days UB=16	7-days UB = 17 & Laziness
Std block approved	283	282	284	282
Std request approved	276	276	275	277
Pref request approved	46	47	46	46
Capacity slack	253.5414	254.9214	252.0214	251.9914
Total slack std-any	295.0	252.0	277.0	289.0
Positive slack std-any variables	16	14	15	16
Total slack std-pref	2.0	4.0	4.0	4.0
Positive slack std-pref variables	1	2	2	2
Total slack priority	0	0	0	0
Positive slack priority variables	0	0	0	0

Table 4.9: Results of experiments with MIPGap = 0.085% instead of TimeLimit

more for the std-pref (this one is harder to fix with manual corrections), while there is also 1 more unit in the capacity violations. Additionally, in Figures 4.18, 4.19 and 4.20 we see the capacity violations of each case in comparison with the Baseline model and we can conclude that the level of violations is the same, thus there is no significant improvement.

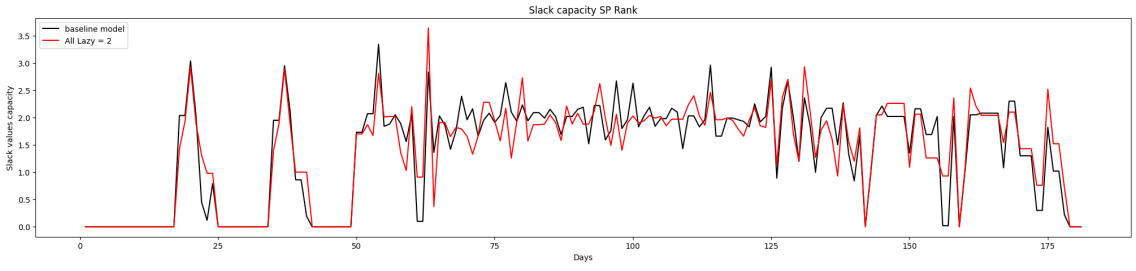

Figure 4.18: MIPGap = 0.085: slack capacity comparison Baseline & All Lazy = 2

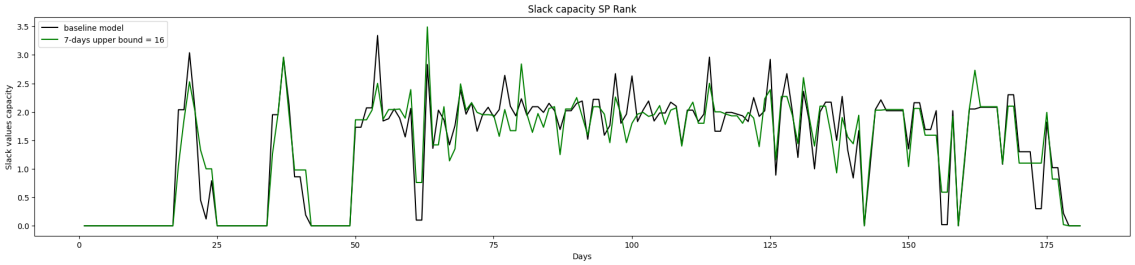

Figure 4.19: MIPGap = 0.085: slack capacity comparison Baseline & 7-days UB = 16

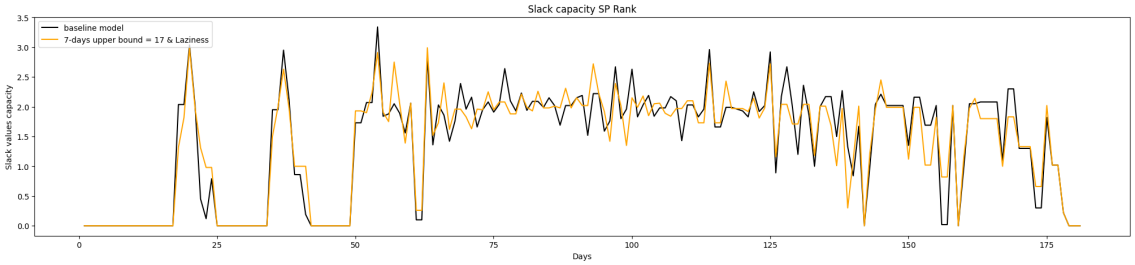

Figure 4.20: MIPGap = 0.085: slack capacity comparison Baseline & 7-days UB = 17 with Laziness

5

Discussion

In this paper, we presented a comparison between various potential improvements applied to the KLM holiday assignment model. This comparison is considered a direct one since all the experiments were conducted under identical settings, with a time limit of 1800 seconds. On top of that, it is essential to highlight that we used the same dataset for all of them, specifically the SP rank dataset with data from the Summer period of 2021. In general, it is a model that cannot reach optimality within a reasonable time and that is why the planners try different scenarios based on produced results. Although it has been indeed tested only for a single dataset, the model (even the baseline) has been built in a specific way that can adapt any dataset that is given, if and only if the given capacity takes into consideration possible changes that happen every year. This is always the case because planners determine different capacities for every year, influenced by various factors such as the number of employees and demand for flights, which do not arise from the model itself.

Undoubtedly, in terms of literature, column generation stands out as the most promising technique for the challenges we face in this model. However, unfortunately, it was not possible to apply it because of the dependency between the columns. On the other hand, we saw that Lazy Constraints can be advantageous. Defining all conflict constraints as Lazy ones (setting "Lazy" attribute to 2) produces the same or slightly better results, especially when planners aim to conduct a 30-minute experiment. However, those results do not imply with less manual corrections, but just a slightly lower final gap. Furthermore, Warm Start is not that promising because Gurobi can find this initial solution in a short time, otherwise the technique would indeed improve the performance. The rest of the techniques were creative ideas that we tried in order to check how the model reacts, and

the 7-days constraint, indeed, was an interesting approach. This type of constraint allowed us to reduce the feasible region by introducing an extra restriction while still maintaining the same level of daily violations.

In general, the business rules (CLA rules) are complicated, and this leads to such a complex MIP formulation that the discussed techniques cannot affect the result as is demonstrated in this paper. Having conducted several experiments and after checking the log output of Gurobi in detail, the only clear impact lies in the convergence process. The convergence (of the Final Gap) in the Baseline model is faster within the first few seconds but it subsequently slows down. Upon applying optimization techniques, such as lazy constraints, there is an acceleration in convergence after approximately 5 minutes. However, all the cases reach approximately the same Final Gap after 30 minutes, indicating no significant improvement in the performance.

6

Conclusion and Future Work

We conclude that no optimization technique can certainly enhance the performance of the model. While small improvements in the final gap have been achieved throughout our research, none of them can result in a reduced need for manual corrections by the planners afterwards. Moreover, the CLA rules result in a complex formulation of the problem, something that is shown in the constraints. There are too many trade-offs that have a crucial impact on the objective function and the solver cannot handle all of them simultaneously. It might seem easy to find the appropriate balance between assigning an employee and causing violations in the conflict constraints, but at the same time, there is a corresponding trade-off between the conflict and capacity violations. And on top of that, the reward for approving a preferred request is double the one for the standard request, but it is the one that causes the standard-preference and priority conflicts. These two types of conflict constraints have the highest penalty for potential violations.

KLM is planning to upgrade the Gurobi cluster capacity eventually, something that will improve model performance stability. In this paper, all the presented results corresponded to the best experiments conducted for each case, certifying that the outcome remained affected by the load of work in the cluster. The lower the load of work in the cluster, the lower the final gap we obtain for each experiment. That is why, when the cluster gets updated, KLM can try the 3 proposed techniques with more extensive experiments. We did not conduct longer experiments (setting higher the `TimeLimit`) in this paper because with the current cluster, there would definitely be some points where CPU usage would reach very high levels (above 80% there is a significant impact on the performance) and have an impact on the convergence and the performance of the model. In those cases, the result would not be realistic. Extending that, in Section 4.7 it is proven that different

experiments with the same final gap yield almost the same solution without considering the time invested in the process.

No matter what, there is a crucial issue with the allotted holiday capacity. The planners try to keep the allotted holiday capacity low in periods where a large number of crew members is needed for production. Based on that, the model should aim not only to assign employees to holidays in a fair way but also to help identify which days require higher capacity. These identifications are made because of the soft capacity constraints, where some violations are allowed. On the other hand, the CLA rules have such a high impact on performance, and crew members can give so many different preferences for holidays that there are too many nodes to be explored. Between the capacity and conflict constraints, there is a trade-off related to the total violations. It would be much easier if one of those factors could be relaxed. A slightly higher level of capacity would have a great impact on the performance since there are max 2-3 capacity violations per day. The result would still be insightful for the planners, as long as the capacity violations would still exist. That way, the model would possibly be solved to optimality within a reasonable time and there would be enough flexibility to experiment with the values of the parameters and Gurobi's configuration, as long as the trade-off between rewards for assignments and penalties for violations. Thus, much more effective support would be provided to the planners.

KLM could also work in the future on reformulating the problem by rephrasing its decision variables. It is not that practical to have a decision variable for the starting day of holidays. Further research can be made on the reformulation in order to figure out how to remove the dependency between the columns and then apply column generation, although it is hard to achieve this since it is related to the constraints which cannot be removed. But it is worth trying because in that case, although the number of decision variables would dramatically increase, column generation would deal with it by finding only the necessary ones and making the convergence faster. Additionally, a decision variable for the starting date of holidays requires a lot of calculations in the capacity and conflict constraints, something that is computationally expensive since they cannot be done in the preprocessing part. Thus, this idea involves another trade-off between the amount of calculation and the number of decision variables, which corresponds to the research question: "Would it be worth it to increase the number of decision variables to decrease the amount of calculation?" The key to the successful answer to this question could be column generation.

The holiday assignment model of KLM gives sufficient satisfaction regarding the fairness towards the employees. Nevertheless, this research suggests that optimizing a bit the options of requests might lead to a solution that is closer to optimality. Since cabin crew members are assigned every year to a different block, it is obvious that they must have a chance to request holidays in another block (preferred block). However, the CLA rule that lets crew members select an entire preferred block results in a very large number of nodes that Gurobi has to explore, and this is hard to do within a reasonable time. Three possible dates as preferred ones, while also taking into account the "hit" day range of +/- 3 days, are more than enough for this type of request. KLM should do some research on how to simplify the CLA rules without reducing the flexibility for the crew, and in such a way that would simplify the assignment process. If this is successful, then an optimal solution would be reached within a reasonable time, which would be at the same time unquestionably fair.

References

[1] Achterberg, T. (2019). What's new in Gurobi 9.0. Webinar Talk url: https://www. gurobi. com/wp-content/uploads/2019/12/Gurobi-90-Overview-Webinar-Slides-1. pdf. 5

[2] Anbil, R., Forrest, J. J., Pulleyblank, W. R. (1998). Column generation and the airline crew pairing problem. Documenta Mathematica, 3(1), 677. 9

[3] Asghari, M., Fathollahi-Fard, A. M., Mirzapour Al-E-Hashem, S. M. J., Dulebenets, M. A. (2022). Transformation and linearization techniques in optimization: A state-of-the-art survey. Mathematics, 10(2), 283. 26

[4] Benayoun, R., De Montgolfier, J., Tergny, J., Laritchev, O. (1971). Linear programming with multiple objective functions: Step method (STEM). Mathematical programming, 1(1), 366-375. 9

[5] Bourhis, P., Reutter, J. L., Suárez, F., Vrgoč, D. (2017, May). JSON: data model, query languages and schema specification. In Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems (pp. 123-135). 5

[6] Brunner, J. O., Edenharter, G. M. (2011). Long term staff scheduling of physicians with different experience levels in hospitals using column generation. Health care management science, 14, 189-202 8

[7] Cheang, B., Li, H., Lim, A., Rodrigues, B. (2003). Nurse rostering problems—a bibliographic survey. European journal of operational research, 151(3), 447-460. 7

[8] Cornuéjols, G. (2007). Revival of the Gomory cuts in the 1990's. Annals of Operations Research, 149(1), 63-66. 10

REFERENCES

[9] De Givry, S., Katsirelos, G. (2017). Clique cuts in weighted constraint satisfaction. In Principles and Practice of Constraint Programming: 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28–September 1, 2017, Proceedings 23 (pp. 97-113). Springer International Publishing. 10

[10] Documentation - Gurobi Optimizaiton. (2022, August 24). Gurobi Optimization. https://www.gurobi.com/documentation/current/refman/lazy.html 26

[11] Documentation - Gurobi Optimization. (2022, August 24). Gurobi Optimization. https://www.gurobi.com/documentation/current/refman/parameters.html 40

[12] Fisher, M. L. (1976). A dual algorithm for the one-machine scheduling problem. Mathematical programming, 11(1), 229-251. 32

[13] Freund, R. M. (1991). A potential-function reduction algorithm for solving a linear program directly from an infeasible "warm start". Mathematical Programming, 52(1-3), 441-466. 9

[14] Gennert, M. A., Yuille, A. L. (1988, December). Determining the optimal weights in multiple objective function optimization. In ICCV (pp. 87-89). 9

[15] Gomory, R. E. (1960). An algorithm for the mixed integer problem (pp. 1-6). California: Rand Corporation. 10

[16] Hafer, L. (2006, July). Tightening 'Big M'Constraints. In DIMACS Workshop on COIN-OR (pp. 17-20). 17

[17] Haessler, R. W., Sweeney, P. E. (1991). Cutting stock problems and solution procedures. European Journal of Operational Research, 54(2), 141-150. 8

[18] Kerrigan, E. C., Maciejowski, J. M. (2000). Soft constraints and exact penalty functions in model predictive control. 8

[19] Kharraziha, H., Ozana, M., Spjuth, S. (2003). Large scale crew rostering. Carmen Research and Technology Report CRTR-0305, Carmen Systems AB, Gothenburg, Sweden, 47. 7

[20] Klabjan, D. (2005). Large-scale models in the airline industry. In Column generation (pp. 163-195). Boston, MA: Springer US. 9

[21] Lübbecke, M. E. (2010). Column generation. Wiley encyclopedia of operations research and management science. Wiley, New York, 1-14. 30

[22] Muts, P., Bruche, S., Nowak, I., Wu, O., Hendrix, E. M., Tsatsaronis, G. (2021). A column generation algorithm for solving energy system planning problems. Optimization and Engineering, 1-35. 8

[23] Pearce, R. H. (2019). Towards a general formulation of lazy constraints. 9, 25

[24] Pearce, R. H., Tyler, A., Forbes, M. (2016). Column Generation and Lazy constraints for solving the Liner Ship Fleet Repositioning Problem with cargo flows. arXiv preprint arXiv:1603.02384. 8

[25] Pedroso, J. P., Rais, A., Kubo, M., Murama, M. (2017). Mathematical Optimization Documentation. 8

[26] Qu, R., He, F. (2008, December). A hybrid constraint programming approach for nurse rostering problems. In International Conference on Innovative Techniques and Applications of Artificial Intelligence (pp. 211-224). London: Springer London. 7

[27] Saemi, S., Komijan, A. R., Tavakkoli-Moghaddam, R., Fallah, M. (2021). A new mathematical model to cover crew pairing and rostering problems simultaneously. Journal of Engineering Research, 9(2). 7

[28] Salehipour, A. (2020). An algorithm for single-and multiple-runway aircraft landing problem. Mathematics and Computers in Simulation, 175, 179-191. 10

[29] Shao, K., Fan, W., Yang, Z., Yang, S., Pardalos, P. M. (2021). A column generation approach for patient scheduling with setup time and deteriorating treatment duration. Operational Research, 1-32. 8

[30] Smith, A. E., Coit, D. W., Baeck, T., Fogel, D., Michalewicz, Z. (1997). Penalty functions. Handbook of evolutionary computation, 97(1), C5. 8

[31] Viegas, R. D., Azevedo, F. (2009). Lazy constraint imposing for improving the path constraint. Electronic Notes in Theoretical Computer Science, 253(4), 113-128. 9

[32] Wilhelm, W. E. (2001). A technical review of column generation in integer programming. Optimization and Engineering, 2, 159-200. 8

[33] Yildirim, E. A., Wright, S. J. (2002). Warm-start strategies in interior-point methods for linear programming. SIAM Journal on Optimization, 12(3), 782-810. 9

Appendix

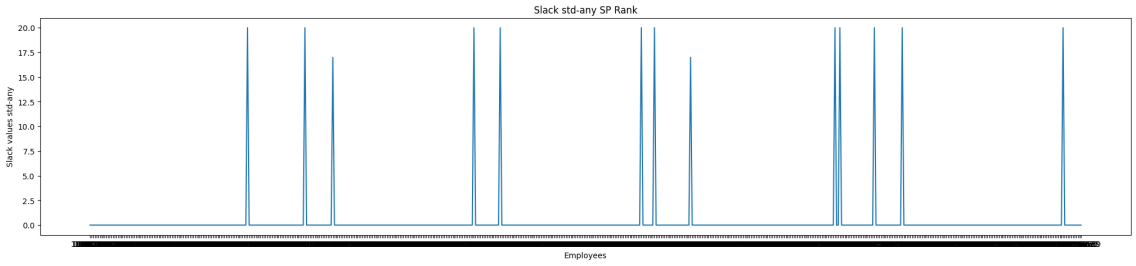

Figure 6.1: SP rank - Current ILP - standard-any conflicts

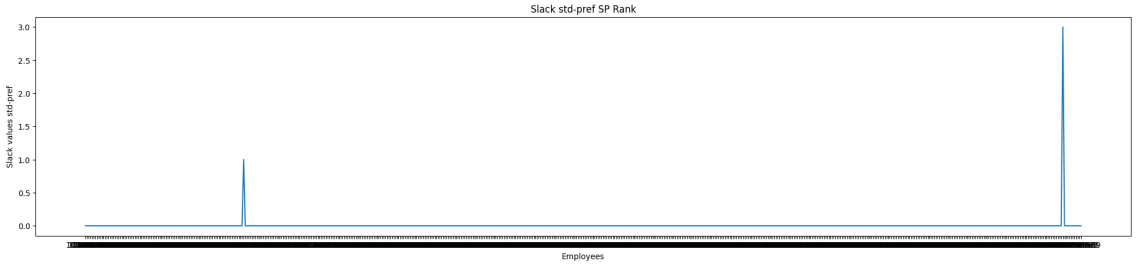

Figure 6.2: SP rank - Current ILP - standard-preference conflicts

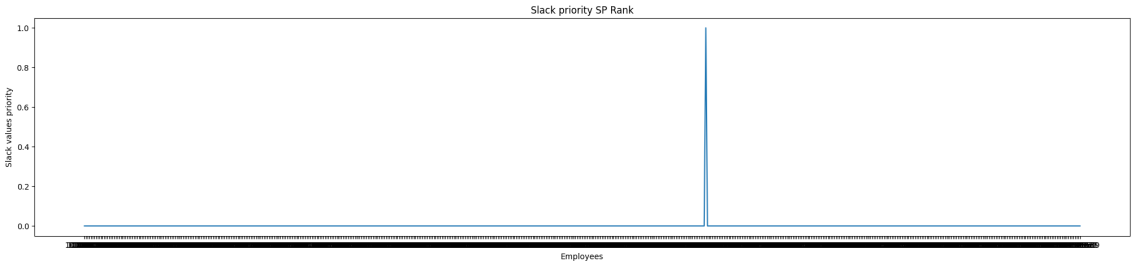

Figure 6.3: SP rank - Current ILP - priority conflicts

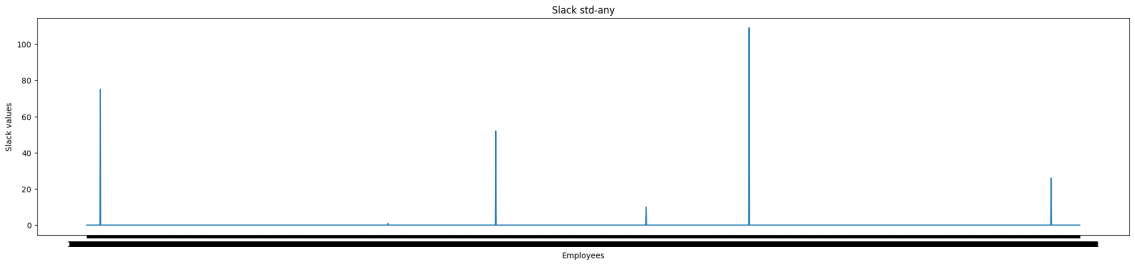

Figure 6.4: BC rank - Current ILP - standard-any conflicts

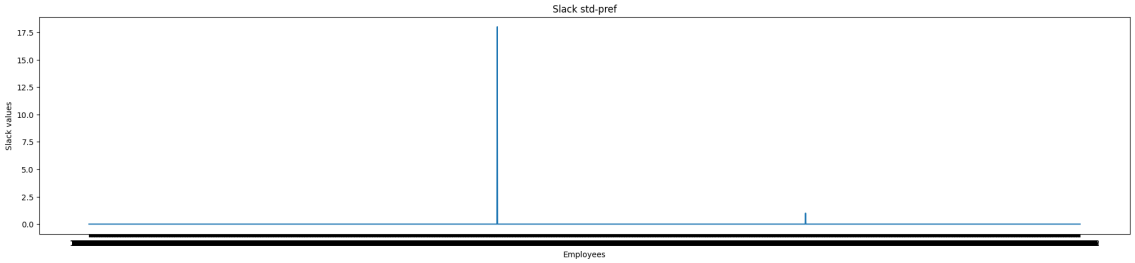

Figure 6.5: BC rank - Current ILP - standard-preference conflicts

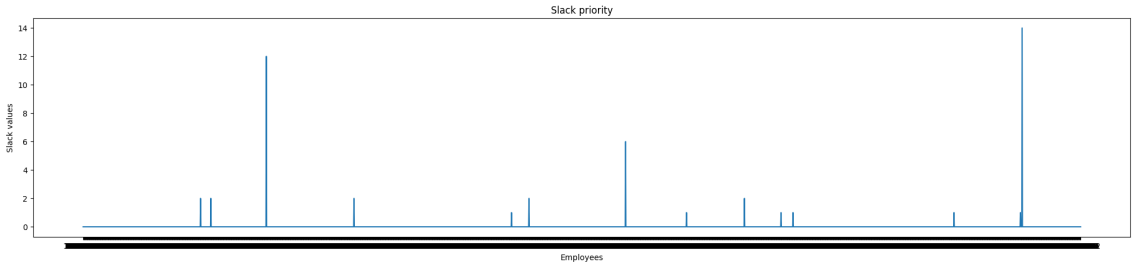

Figure 6.6: BC rank - Current ILP - priority conflicts

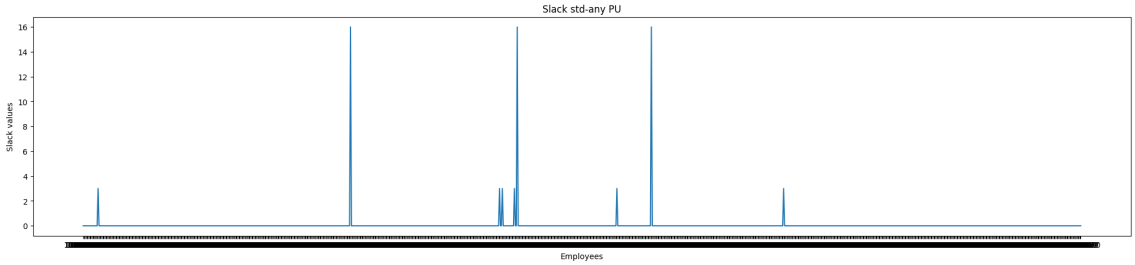

Figure 6.7: PU rank - Current ILP - standard-any conflicts

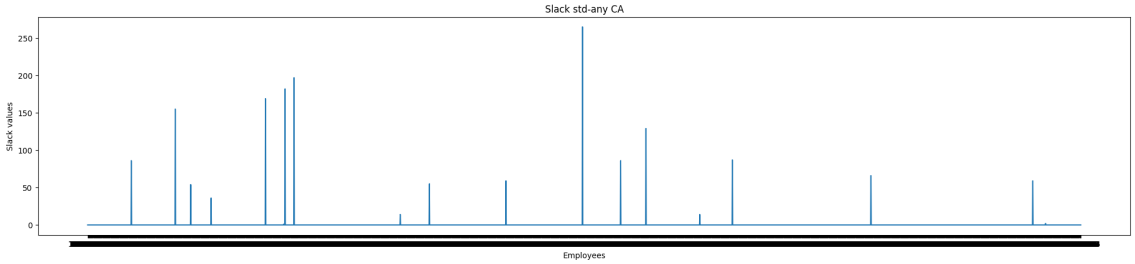

Figure 6.8: CA rank - Current ILP - standard-any conflicts

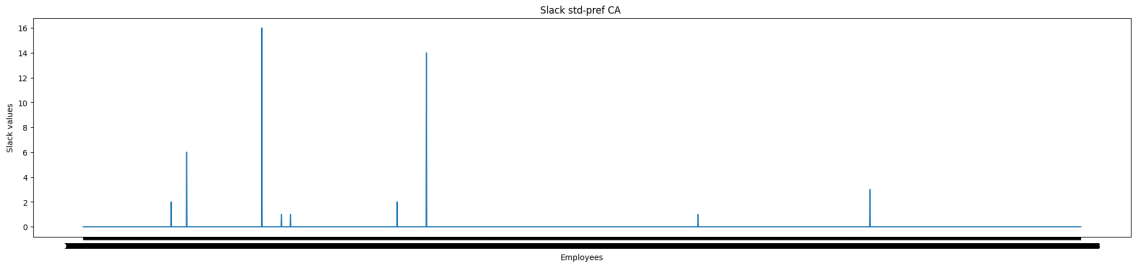

Figure 6.9: CA rank - Current ILP - standard-preference conflicts

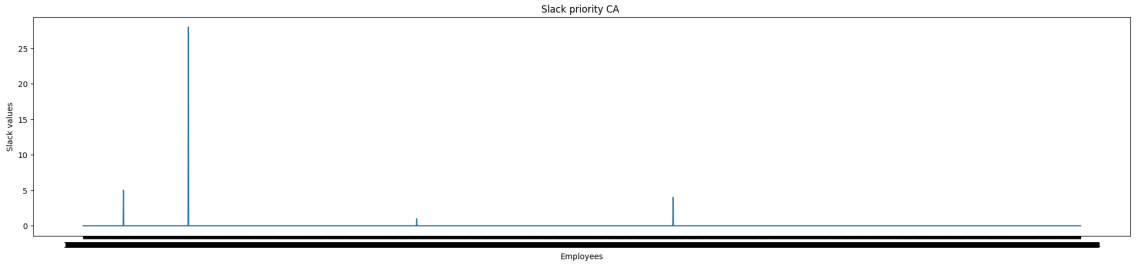

Figure 6.10: CA rank - Current ILP - priority conflicts