Optimization Approaches for Solving Large-Scale Personnel Scheduling Problems

by

Junhong Guo

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Industrial and Operations Engineering)
in the University of Michigan
2021

Junhong Guo

jhguo@umich.edu

ORCID iD: 0000-0002-6531-6001

This dissertation is dedicated to people all over the world fighting COVID-19 and to people promoting world peace.

ACKNOWLEDGEMENTS

I want to first express my gratitude to my advisor and dissertation committee chair: Professor Amy Cohn. Thank you for providing me with the opportunity to pursue my PhD at UM and introducing me to two wonderful families: IOE and CHEPS. I am grateful for your help, support, and patience during this journey. PhD study is never easy and there were so many challenges, but you have made the past four years also truly enjoyable for me. I want to thank you for your academic guidance, for caring about my mental well-being at all times, and for your support during the toughest moments I experienced. Without you, I would not have been able to accomplish any of my research and reach this point. You have generously shared a lot of insights on both academia and industry with me, and I have learned a lot from you, which will help me in paving my own career path. A mentor of life. I could not be more grateful.

Besides Amy, I would like to thank each member on my committee for their guidance and help during my PhD journey. Professor Marina Epelman, thank you for working with me on the crash project for staffing the hospitals at the early stage of the pandemic. I truly enjoyed our collaboration and I will never forget this valuable experience. Professor Viswanath Nagarajan, thank you for patiently answering all my questions and for generously helping me find the supplementary materials and journal papers I requested at all times. Professor John Silberholz, thank you for meeting with me on several occasions and sharing your insights on healthcare, engineering, optimization applications, and so much more with me. Thank you all for your continued guidance and great comments on my dissertation, which has prepared me to reach this point.

I want to thank my mom and my dad for accompanying me going through this process. Studying abroad is hard. I encountered many challenges and, particularly during the past couple of years, I experienced several really tough periods. I have successfully overcome all of them, which would not be possible without your love and support. I also want to thank my grandparents, my aunts and uncles, my siblings, and my whole extended family who witnessed and celebrated every achievement I have made throughout my life.

Next, I want to express my gratitude to all of the faculty and staff who have promoted me in all different angles and helped me move forward step by step along my educational journey. It would take a dozen pages to write my appreciation to each of you. Shane, thank you for advising me on my intriguing master's degree project at Cornell and introducing me to the fantastic world of optimization and healthcare. Billy, thank you for onboarding me during my first day at CHEPS. Besides work, I have learned a lot from you on medical education, football, and UM's athletics system, which all have helped me (an international student) get more familiar with US culture; I really appreciate it. Liz, I cannot be more thankful for your editorial help on my journal paper submissions, presentation slides, and symposium posters. Particularly, I want to thank you for reviewing and proofreading this dissertation, which was really helpful. Gene, every time seeing your warm smiley face when getting to the CHEPS office would truly make my day. Also, thank you for sharing your tennis stories with me, and as promised, I will find a chance and go to the US Open when this pandemic is over and share my experience with you! Tina, thank you for your continued help on the complicated paperwork and for always being so patient and nice with me. Thank you again to all of CHEPS and the IOE staff for generously supporting me during my time at UM.

A special thank you to all of the students who have helped me with my research and its dissemination. I would like to thank all students in the aviation team for your support on our crew pairing project. Especially Theodore Endresen, thank you for providing me with so many creative and fantastic ideas to enhance our approach, carrying out all batches of different experiments, and reviewing and revising the paper back and forth with me. We could not achieve those exciting results without your enthusiasm, productiveness, and dedication. I also want to thank both current and previous students on the block team for supporting my research on resident scheduling. Shamit, Shraddha, Ziqi, and Jiaqi, thank you for your help on conducting the experiments and structuring, visualizing, and analyzing the results. Your hard work has provided me with a great amount of insight on the directions to further dive into to improve my research.

Next thank you to my mentors Karmel Shehadeh and Donald Richardson. Thank you for your great advice not only on academic study and research, but also on life and career development. I could not be more grateful for your supporting my transition to the PhD study and on my work during the past four years. I also want to thank Weiyu, Adam, Qi, Sentao, and Yiling for helping me go through each check-point of the PhD process, generously sharing many great resources with me, and chatting with me every now and then on a wide range of interesting topics. A big thank you also goes to my amazing Year-2017 Cohort. I will always keep those precious memories in my mind: that we studied super hard for homework and exams in the “Dungeon” and we celebrated every milestone we had achieved along the journey together.

I would like to thank all of my friends for helping me keep a good work-life balance. Especially, I want to say thank you to Luze, my officemate and best buddy. I really enjoyed the time we spent together studying for coursework, discussing research questions, playing basketball, tennis, and video games, watching NBA games, practicing for our driver’s licenses, and so on. There are so many wonderful moments with you, which have made the past four years one of the most enjoyable periods through my life. Next thank you to Duoxi for accompanying me every time I went to Boston and cheering with me in TD Garden for Celtics. I want to also thank Chenyang, Ziyang, Yi, Xingyu, Yukun, and many other my high school friends for your help and support. I look forward to having reunions with you all again in the near future. Thank you to all my Pokemon buddies, Weitao, Zheng, Yuanzhi, Sicen, Guanglong, Xiaoer, Hongling, Xinyan, Xinjing, and Dejiu. We have attended so many events together, and I will keep those amazing memories in my mind, forever.

Lastly, I gratefully thank Seth Bonder foundation, IOE Department, and CHEPS for their generous financial support for my PhD study and research. Especially, I want to thank Mrs. Merrill Bonder for her continued support and encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Table

ABSTRACT

Personnel scheduling is one of the most critical components in logistical planning for many practical areas, particularly in transportation, public services, and clinical operations. Because manpower is both an expensive and scarce resource, even a tiny improvement in utilization can provide huge expense savings for businesses. Additionally, a slightly better assignment schedule of the involved professionals can significantly increase their work satisfaction, which can in turn greatly improve the quality of the services customers or patients receive. However, practical personnel scheduling problems (PSPs) are hard to solve because modeling all of the complicated and nuanced requirements and rules is challenging. Moreover, since an iterative construction process may be necessary for handling the multiple-criteria or ill-defined objective nature of many PSPs, the model is expected to be solved in a short time while providing high-quality solutions, despite its large size and complexity. In this dissertation, we propose new models and solution approaches to address these challenges.

We study in total three real-world PSPs. We first consider the crew pairing construction for a cargo airline. Each crew pairing is a sequence of flights assigned to a specific line/bid crew to operate in practice. Unlike traditional passenger aviation, due to the cargo airline's underlying network, each crew pairing will specify a complete flying schedule for the assigned crew over the entire planning horizon. Consequently, an extra and unique set of requirements must be incorporated into the construction process. We solve the problem using a delayed column generation framework. We develop a restricted shortest path model to incorporate the entire set of complicated requirements simultaneously and solve it using a labeling algorithm accelerated by a handful of proposed strategies. Computational experiments show that our approach can solve the crew pairing problem in a short time, while almost always delivering an optimal solution.

Second, we consider an extension of the previous cargo crew scheduling problem, where a “break” is allowed to take place in the “middle” of each crew pairing. This break feature, working as a special type of conventional deadheading, is expected to significantly increase the flight coverage for practical deployment. However, incorporating this feature will result in an extremely dense underlying network, which introduces new computational challenges. To address this issue, we propose a bidirectional labeling based arc selection approach, which only needs to work on a tiny sub-network each time but can still guarantee the exactness of the delayed column generation process. We demonstrate through real-world instances that our proposed approach can solve this relaxed problem extension in a very short time and the resulting flight coverage will increase by over 30%.

Finally, we study a medical resident annual block scheduling problem. We need to assign residents to perform services at different clinical units during each time period across the academic year so that the residents receive appropriate training while the hospital gets staffed sufficiently. We propose a two-stage partial fixing solution framework to address the long runtime issue caused by traditional approaches. A network-based model is also developed to provide a high-quality service selection to initiate this two-stage framework. Experiments using inputs from our clinical collaborator show that our approach can speed up the schedule construction at least 5 times for all instances and even over 100 times for some huge-size ones compared to a widely-used traditional approach.

CHAPTER 1

Introduction

Personnel scheduling problems (PSPs) are frequently encountered in a wide range of practical areas, e.g., healthcare, transportation, protection & emergency, manufacturing, and military services. Considering the high expense in terms of the human resource as well as the fact that the decisions on this resource allocation typically need to be made on a fairly regular basis, high-quality solutions are extremely desirable to assist decision-makers with better resource management. Even a tiny improvement can bring substantial benefits in practice.

Generally, PSPs consist of assigning a sequence of tasks to each person involved in the schedule across a given planning horizon. However, the number of possible assignments plus nuanced constraints from practical operation restrictions make these scheduling problems extremely difficult to solve. Multi-criteria, ill-defined objectives introduced by preferences from different stakeholders also add challenges. To address these challenges, a few iterations of “result, review, and revise” are typically required before the schedule can be finalized. That is, the decision-makers will review the current solution, provide feedback and comments, and, based on that feedback, modifications to the model are made to generate an updated version, and repeat. Since the planning phase is expected to be done within a short period to ensure sufficient time for schedule deployment and the associated administrative logistics, the construction time for obtaining high-quality solutions would in general be the most critical metric for solving practical PSPs.

Many traditional methods in the literature are developed based on idealized assumptions, and thus cannot be directly applied to solve practical scenarios. For example, they only consider and enforce basic, fundamental restrictions or they assume the underlying structure of the problem holds specific desirable properties. Additionally, many of those methods have scalability and/or flexibility issues, which prevent them from incorporating additional requirements and handling problem variations.

In this dissertation, we develop new formulations and solution approaches for large-scale PSPs that are challenging to solve by traditional methods, with an emphasis on achieving the following 3 goals:

- **Efficiency**: Speed is the core. As mentioned above, a single instance often needs to be solved iteratively as extra requirements or modifications may be proposed by decision-makers after solution review.
- **Solution Quality**: The model should be developed with mechanisms to guarantee the solution quality so that the resulting solutions can be implemented in practice while also being satisfactory to decision-makers.
- **Flexibility**: The model should be flexible and it should be easy to incorporate additional requirements, without a significant modification to the formulation itself.

We mainly focus on solving PSPs in two fields — crew pairing/scheduling for cargo aviation and resident scheduling for medical trainees. Air cargo is growing significantly faster than passenger aviation and is forecast to maintain a stable growth rate over the coming 20 years. Given this, cargo airlines nowadays are facing the challenge of accommodating market demand while controlling their operating costs. In particular, contracting crews introduces the second-highest expense (right after fuel) to the operations of most airlines' business, and therefore the quality of crew schedules can be largely decisive to the profitability of an airline. Residency programs in medical school prepare doctors in different fields to become attending physicians after earning their M.D. degrees. Resident scheduling is one of the most crucial logistical elements of clinical education. It ensures residents receive appropriate training, while different units of the hospital where they are undergoing their training are staffed sufficiently to provide care to patients. Administrative preferences and resident requests for vacations and electives also need to be considered in the schedule construction, and a high-quality schedule helps avoid burnout, maintain a good work-life balance, and ensure residents can pursue the specialties they are interested in, which is highly desirable to all stakeholders.

We propose new models and solution approaches in this dissertation to deal with PSPs in these two fields so that high-quality solutions can be more efficiently produced. We work closely with our industrial collaborators to ensure our models and algorithms are capable of addressing representative scenarios in these important fields. Consequently, the high-quality solutions generated can assist our collaborators with much better resource management and operation cost control, and can also implicitly improve the satisfaction of people involved in the planning phase during the actual operations. Moreover, our models and approaches are flexible enough to easily incorporate potential modifications in the future.

Because crew scheduling in transportation and healthcare provider scheduling in the clinical environment are the two most representative applications in PSPs, our proposed formulations and methods can be generalized to solve a wider range of complicated large-scale PSPs. Thus, our research has the potential to promote the development of healthcare delivery and scarce resource utilization in many other areas.

The remainder of this dissertation is organized as follows. In Chapter 2, we consider the problem of generating high-quality crew pairings for a cargo airline, where each crew pairing is a sequence of flights satisfying specific requirements like labor regulations, and will be assigned to a single crew to carry out. In Chapter 3, we work on an extension of the cargo crew pairing problem where a potential break period is additionally allowed to take place in each crew pairing, in order to increase flight coverage. In Chapter 4, we study an annual block scheduling problem for medical residents, where decisions on the assignments of residents to different clinical units for each time period (a.k.a. block) during the academic year need to be made, while a huge set of rules and requirements must be satisfied. Lastly, in Chapter 5, we summarize our research and provide insights for future work. A detailed description of the contents of Chapter 2, 3, and 4 is elaborated below.

Chapter 2: Unlike traditional passenger crew scheduling or pairing problems, the nature of the cargo network that we consider is such that to ensure all pairings are of reasonable length and workload, we are not able to cover all of the flights in the planning horizon. We, therefore, propose a maximum weighted set packing problem rather than a set partitioning/covering problem to model this problem. Due to the very large number of viable pairings, the huge size of the model requires us to use a delayed column generation approach rather than solving it explicitly. We formulate the pricing problem of the delayed column generation framework as a shortest path problem with resource constraints (SPPRC) to dynamically generate crew pairings — an approach commonly used for routing and scheduling problems. A labeling algorithm is then developed to solve this SPPRC formulation, and several modifications are proposed to further improve its computational performance.

Compared with passenger aviation, crew scheduling and pairing construction for cargo aviation have received significantly less attention. To the best of our knowledge, almost all work on air cargo scheduling in the literature has focused on flight schedule design, airport selection, fleet assignment, aircraft routing, and cargo routing, where crew scheduling and the corresponding restrictions were not taken into account (Yan et al., 2006; Li et al., 2007; Tang et al., 2008; Derigs et al., 2009; Derigs and Friederichs, 2013). The research in this chapter, therefore, contributes to filling this gap.

What makes the cargo crew pairing problem unique and difficult to solve is the nature of its underlying network, which largely consists of long-haul international flights and lacks repeating daily patterns and opportunities for quick turns. Consequently, each valid crew pairing has to directly specify the complete flying schedule, spanning approximately half of a month, for the assigned crew during the entire planning horizon, instead of one to three days like traditional passenger crew pairings. This introduces an extra set of unique and complex regulations and agreements to our crew pairing problem, as the conventional subsequent crew rostering step must now be partially accomplished by this step in the whole crew schedule planning phase. We present modeling and algorithmic approaches to overcome these new challenges in this chapter and demonstrate our results using real-world datasets.

Chapter 3: Due to the nature of the cargo network, we frequently cannot achieve sufficient flight coverage, even by the theoretically best crew pairing solutions. This low coverage is unacceptable for practical deployment since operating the remaining uncovered flights requires using reserved, extra labor resources, which will be much more expensive. To address this issue, the cargo airline allows crews to fly home commercially once during their assigned flying schedule, which requires us to model a potential "break" in the "middle" of the crew pairings. By incorporating this break feature, each crew pairing can then consist of up to two flying segments (which actually re-defines the concept of crew pairings), where the crew is allowed to head to the base and stay home for some time after the first segment and before starting the second one. Consequently, this break feature, as a special deadhead, relaxes the leg consistency requirement on crew pairings, and therefore can significantly boost flight coverage.

The proposed break feature has commonalities with the maintenance requirement in airline fleet assignment and rolling stock scheduling as well as the day-off feature in driver scheduling in public transportation and crew scheduling in railways, as they all serve as the "hole(s)" separating duty segments in the complete schedule. However, having a break in our crew pairing is an auxiliary option for boosting the underlying network connectivity, instead of a mandatory requirement that each crew must perform. In addition, unlike the maintenance, which is often assumed to be done overnight only at some predetermined spots, there is no such limitation to our break. Instead, we consider the restriction on the relative position of the break within the whole schedule, which is generally not taken into account for the maintenance and the day-off features. Given these, the well-studied approaches proposed in the literature for dealing with the maintenance and the day-off requirements, e.g., the layered time-space network flow model by Şahin and Yüceoğlu (2011), the rotation-tour time-space network by Liang and Chaovalitwongse (2013), the hypergraph representation by Borndörfer et al. (2016), the two-stage heuristic approach by Zhong et al. (2019), etc., are not applicable to our problem.

Moreover, incorporating this break feature introduces a great number of arcs to the underlying network. This resulting extremely dense network makes the traditional solution approach used in Chapter 2 inadequate for solving real-world instances, which introduces new computational challenges. We present a bidirectional labeling based arc selection approach, so that the majority of arcs can be temporarily removed from the network, and thus only a tiny sub-network needs to be considered for each column generation iteration. We prove that this arc selection is an exact approach and solving the pricing problem of the delayed column generation framework based on the pruned, tiny sub-network is equivalent to solving the original one. We verify the effectiveness of our approach by computational tests on real-world instances, and furthermore demonstrate that it can totally outperform an intuitive partial pricing heuristic, in terms of both runtime and solution quality.

Chapter 4: We move from cargo crew pairing and scheduling to the annual block schedule construction for medical residents in this chapter. Although the background of the problems is different, they have many commonalities with each other in features like the huge number of possible combinations by the large-scale nature, the complicated requirements and rule structures, and so on, which introduce significant difficulties to the schedule construction. More specifically, to build a valid block schedule for each resident, besides many side requirements and rules, we need to coordinate a large set of educational requirements, which ensure residents are receiving appropriate training through performing services at different clinical units, and a large set of coverage requirements, which guarantee all units (services) in the hospital are staffed sufficiently. The huge size and complexity result in an extremely difficult combinatorial problem to solve to produce a feasible resident schedule. Since a "result, review, and revise" iterative process is required for decision-makers to gradually adjust requirements to meet actual needs and to incorporate preferences from both residents and administration, a feasible schedule for each round, satisfying all requirements and rules at that moment, is expected to be obtained within a very short time.

Compared with shift scheduling (Sherali et al., 2002; Cohn et al., 2009; Topaloglu and Ozkarahan, 2011; Güler et al., 2013), constructing resident block schedules has received significantly less attention. Furthermore, we have to consider the schedule construction for multiple residency programs simultaneously because there is a hybrid program that shares responsibility for covering units under other programs in the specific medical school we collaborate with. As a result, the decomposition by constructing the schedule program by program is not applicable, and the size of our problem is much larger than problems typically considered in the literature (Franz and Miller, 1993; Bard et al., 2016, 2017).

A key contribution of this work is in proposing a novel solution framework to address the computational challenge of solving this large-scale resident scheduling problem, as conventional, widely-used approaches like branch-and-cut by a mixed integer program (MIP) solver are shown to be not sufficient in practice. Although there is not a natural two-stage decision-making structure, we intentionally partition our decision process into two stages. We first consider making the allocation of residents for a small number of "picky" services, whose associated constraints already make these assignments highly restricted. Then, we try completing the remaining pieces of the puzzle after partially fixing these assignments. We develop several cut generation mechanisms to prune off the current unacceptable fixing once infeasibility arises, which therefore ensures this solution framework is an exact method. We've applied this solution framework to practical instances, and the results show that our cut generation mechanisms can provide a significantly more robust performance than an intuitive method, while our approach can largely outperform the traditional approaches that are commonly proposed in the literature.

In order to ensure a desirable performance of this solution framework, we develop a network structure to quantitatively represent the underlying relationships among different services in terms of the requirements on resident resource allocation. Based on this, we formulate and solve a system of linear equations to identify an ideal set of services to be fixed regarding their assignments in the proposed two-stage approach. This linear equations system balances our preferences for selecting services based on tradeoffs between their individual flexibility in terms of resident allocation requirements and restrictiveness of their interactions with other services. We demonstrate through experiments on real-world instances that this network-based method can significantly reduce the total runtime required by our proposed two-stage model to solve the original, entire scheduling problem.

In summary, we consider solving three practical large-scale personnel scheduling problems arising from operations in two important fields. We propose new modeling and solution approaches to address a wide range of challenges, including the incorporation of unique requirements, solution quality improvement, the long runtime bottleneck, the unstable computational performance, and so on, which thus advances the corresponding science frontier. In addition, we want to point out that the flexibility of all our approaches enables them to handle problem variations with different or new requirements and objectives, and allows them to be generalized and applied to other applications.

CHAPTER 2

A Delayed Column Generation Approach for Cargo Crew Pairing Construction

2.1 Introduction

In this chapter, we consider a cargo airline crew pairing problem in which the key decisions are to determine how to sequence flights to be assigned to cockpit crews over a planning horizon. In this section, we provide an introduction to the air cargo industry and a brief description of the crew scheduling problem for a specific cargo airline where the crew pairing is the most critical part.

2.1.1 The Cargo Aviation Industry

In the United States, Europe, and the Asia-Pacific region, air cargo volume has grown on average 50% faster than air passenger volume from 1995 to 2004 (Wong et al., 2009). According to Boeing (2018), global air cargo traffic is forecast to grow a robust 4.2% per year over the next 20 years, the revenue ton-kilometers (RTKs) will more than double from 256 billion in 2017 to 584 billion in 2037, and the number of freighter airplanes will grow by more than 70% in total. Given this rapid growth, cargo airlines nowadays are facing the challenge of accommodating this market demand while controlling their operating costs.

A cargo airline accepts requests for goods delivery, from one location to another, from customers including logistic companies, manufacturers, the military, etc. These requests are gathered and further partitioned into different planning horizons, typically a calendar month. The cargo airline needs to determine and schedule all necessary tasks and activities accordingly so that the requests in each planning horizon will be delivered as planned.

The cargo airline that we partnered with has a fleet of airplanes and two types of crews it can contract. *Line/bid crews* are awarded a set schedule for the planning horizon while *reserve crews* operate open, uncovered flights left by incomplete coverage in the planned schedule, schedule disruption, or illness of line crews. Scheduling these resources to carry out the corresponding delivery tasks in the planning horizon is a critical decision-making process that heavily impacts profitability.

The cargo aviation scheduling process, in most cases, can be divided into several phases: schedule design, fleet assignment, aircraft routing, crew scheduling, and cargo routing (Derigs and Friederichs, 2013). Prior to the crew scheduling phase, a set of flights (specified by aircraft number, origin, destination, departure time, and arrival time) will be determined, where scheduled requests in the planning horizon will be distributed and handled across all these flights. Crew scheduling is then assigning each crew to a subset of these flights to operate in practice.

2.1.2 Crew Scheduling in Cargo Aviation

In this chapter, we consider the crew scheduling phase for our partnered cargo airline. More precisely, we focus on constructing the flying schedule for the line crews that the airline contracts with. Due to a large number of nuanced regulations from the Federal Aviation Administration (FAA) as well as many other side constraints (e.g., tied to labor contracts), crew scheduling is a complicated and time-consuming decision process. Since the flight schedule is only set approximately one month ahead, the crew schedules are expected to be constructed in a short time (i.e., four to five days) in order to ensure there is a sufficient amount of time for implementation and deployment. At our partnered airline, crew schedules were previously constructed manually, a process that can be error-prone and time-consuming. Therefore, we set out to develop a more efficient decision support-based approach.

A key component of crew scheduling is in developing a set of *crew pairings*. A crew pairing is a sequence of flights that will be assigned to a single crew to carry out with specific requirements that must be satisfied. For traditional passenger aviation, the crew scheduling problem can be split into two sub-problems — the *crew pairing problem* and the *crew rostering* or *bidline problem* (Gopalakrishnan and Johnson, 2005). The crew pairing problem is to generate a set of valid crew pairings to cover the scheduled flights in the planning horizon (Anbil et al., 1998; Haouari et al., 2019); the rostering or bidline problem is to then construct the schedule for individual crews through concatenating the generated pairings with other activities during the planning horizon (e.g., training and vacations) (Gamache et al., 1998, 1999; Kasirzadeh et al., 2017). The crew schedule construction at our partnered airline follows this two-phase subsequent procedure but differs significantly in both the pairing and rostering steps.

For traditional passenger aviation, the crew pairing problem can usually be further decomposed. Since almost all flights in the planning horizon are repeated daily for the majority of domestic passenger carriers, crew pairings that span on the order of one to three days, starting and ending at the same crew base, are first constructed to cover the daily flights. Then, these pairings are duplicated and dated accordingly with minor modifications to cover the entire planning horizon. However, this decomposition doesn't work for the pairing problem for our partnered cargo airline because our cargo flight network lacks any repeating pattern. Instead, we must consider all of the flights scheduled in the planning horizon at the same time to generate crew pairings. In addition, since the majority of our flights are international and long-haul, each crew pairing for our problem has to correspond to a complete crew schedule (in terms of flying tasks), which spans a much longer time (e.g., a half month), and is ready to be assigned to a contracted crew to carry out. In other words, we construct our crew pairings by partially integrating the conventional crew pairing and crew rostering/bidline phases, as the concatenation of "pairings" will be all set during this step. For the same reasons, the life of a member of our cargo crew is also very different from traditional passenger crew's. The crews in our problem are usually away from home for the duration of much longer pairings — at least 12 days, versus one to three, as in the domestic passenger case, and they often fly commercially to the origin of the first flight in the pairing and home from the destination of the last flight in the pairing.

Since each pairing constructed by the crew pairing phase corresponds to a complete flying schedule during the planning horizon, the rostering phase for our partnered cargo airline is trivial. After all of the crew pairings are created, they would be posted for all of the line crews to view. Then, the crews would select which crew pairing they wanted to fly, taking their training and vacation preferences into account, in order of their seniority as determined through their labor contract.

In this chapter, we mainly focus on solving the crew pairing problem for our partnered cargo airline, as the crew pairing construction is the most critical part, and largely deterministic for profitability for the whole crew scheduling.

2.1.3 Outline

The remainder of the chapter is structured as follows: in Section 2.2, we present a detailed statement of our cargo crew pairing problem. In Section 2.3, we briefly discuss previous work in the literature, particularly on crew scheduling (pairing & rostering). Section 2.4

provides a description of the maximum weighted set packing formulation and the delayed column generation framework that we propose for solving our crew pairing problem. In Section 2.5, we describe the formulation for the pricing problem, and present a conventional approach for solving it, as well as a set of speed-up modifications and improvements. Section 2.6 presents computational experiments' results on real-world instances. Lastly, in Section 2.7, we conclude and provide thoughts for future work.

2.2 Problem Statement

As mentioned previously, for a crew paring to be valid, several requirements must be satisfied. These requirements include basic "laws of physics" (to ensure flights are continuous in space and time), FAA regulations (for flight safety and the health of the crew), and the airline's collective bargaining agreement (for company/worker satisfaction). More specifically, we consider the following seven pairing requirements for our problem:

(1) The origin of a flight should be the same as the destination of its preceding flight, and there must be a minimum time period (e.g., 45 minutes) between two consecutive flights for transition.

(2) After continuously working for up to a maximum period of time (e.g., 17 hours), the crew must have a minimum amount of time for rest (e.g., 10 hours), which is called a *layover*. Each working period between layovers is defined as a *duty period*, and the idle time between two sequential flights is regarded as working time and counted toward the duty rules if it is not sufficient to count as a layover (and this short idle time period is called a *sit-time*).

(3) During each duty period, in addition to the limit on overall time, the cumulative flight

time cannot exceed a specific upper bound (e.g., 12 hours).

(4) If a crew has already had a specific number of duty periods in a row (e.g., 6 consecutive duty periods) without a layover in between which is greater than or equal to a specific lower bound (e.g., 24 hours), then they must have a longer rest period of at least this minimum amount of time (we call this a *day off*), before starting the next duty period.

(5) The time span of the crew pairing cannot exceed a specific upper bound (e.g., 16 days).

(6) The time span of the crew pairing must be greater than or equal to a specific lower bound (e.g., 12 days).

(7) The total cumulative flight time of the pairing must be greater than or equal to a specific lower bound (e.g., 70 hours).

The last two requirements, i.e., the lower bounds on the pairing span and total flight time, need to be explicitly enforced in our context in order to avoid deploying short crew pairings. This is because our crew pairings will correspond to complete crew flying schedules, and the airline must respect the minimum flying hours for the assigned line crews per labor contracts, and wants to keep the workload balanced across different crews for fairness.

In the problem instances that we consider, it is often not possible to construct a set of valid pairings that collectively fully cover all flights. This is because the flight network of the cargo airline lacks opportunities for quick turns, and includes many airports with only a small number of associated flights, which results in a much smaller number of possible flight combinations and a very limited number of valid pairings for covering some specific flights. Given this fact, the cargo airline will use its reserve crews to handle the uncovered flights. However, the cost for a reserve pilot to cover these remaining flights is more expensive both in monetary cost and crew efficiency. Thus, based on the estimated excessive expense of flying each scheduled flight by a reserve crew, the cargo airline's objective is to minimize the total excessive cost, caused by the reserve crew usage.

2.3 Literature Review

Passenger crew scheduling problems have been widely studied over the past 40 years. As already mentioned in Section 2.1.2, a crew scheduling problem is typically divided and solved through two sequential sub-problems. First, a crew pairing problem is solved to generate a valid, unassigned set of crew pairings to cover scheduled fights in the planning horizon (typically a calendar month). Based on these generated pairings, a crew rostering or bidline problem is then solved to form a set of complete schedules for crews to operate in practice, where crew pairings are grouped and concatenated with other personalized activities and tasks, like training and vacations.

Regarding the crew pairing problem, in many cases, flights in a much smaller horizon (e.g., a day or a week) will be included for the pairing generation, despite some explicitly based on the original horizon (e.g., the calendar month) (Erdoğan et al., 2015), depending on the repeating flight pattern deployed by the airline. For example, the majority of research in the literature deals with daily crew pairing problems (Vance et al., 1997; Klabjan et al., 2002; Sandhu and Klabjan, 2007; Dunbar et al., 2012; Shao et al., 2015; Haouari et al., 2019), where the same set of flights are assumed to be flown every day in the planning horizon. Solutions to this problem will collectively cover all flights occurring within the daily schedule. Later, these pairings will be repeated daily across the original, longer horizon (e.g., a week or month). These date-specific pairings will then be used to construct the crew schedules during the subsequent crew rostering/bidline problem. When flight patterns repeat weekly rather than daily, a similar approach may still be used (Lavoie et al., 1988; Yan and Tu, 2002).

The cost associated with pairings is defined in various ways, taking into account factors such as time away from base (TAFB), duty costs, layover costs, and potential for delay/disruption propagation (Lavoie et al., 1988; Barnhart et al., 1994; Desaulniers et al., 1997; Vance et al., 1997; Anbil et al., 1998; Barnhart and Shenoi, 1998; Klabjan et al., 2002; Sandhu and Klabjan, 2007; Dunbar et al., 2012; Shao et al., 2015; Cacchiani and Salazar-González, 2017; Wei and Vaze, 2018; Haouari et al., 2019), as airlines evaluate the operational costs based on different measurements, and aim to achieve different goals from different perspectives.

To mathematically formulate the crew pairing problem, set partitioning formulations and set covering formulations are commonly used, which both select a subset of valid crew pairings with the collectively smallest cost. The set partitioning formulation enforces each flight to be covered exactly once (Anbil et al., 1998; Yao et al., 2005; Weide et al., 2010; Dunbar et al., 2012; Shao et al., 2015; Wei and Vaze, 2018), while the set covering formulation requires all flights to be covered at least once (Lavoie et al., 1988; Barnhart et al., 1994). In the latter case, if a specific flight is covered by more than one pairing, only one of the corresponding assigned crew will actually operate this flight, while the other crews will fly as passengers.

A number of methods and algorithms have been developed to solve the set partitioning / covering formulations. *TRIP*, a local search approach, is one of the first proposed methods for solving the set partitioning formulation for crew pairing problems (Gershkoff, 1989;

Anbil et al., 1992), developed based on *TPACS* (Rubin, 1973). Brute force methods like depth-first-search (DFS) that fully enumerate all feasible pairings have also been applied, where the corresponding formulation is solved explicitly based on the enumeration (Baker and Fisher, 1981; Aggarwal et al., 2018).

Rather than a few thousand valid pairings that exist in the instance in Baker and Fisher (1981), the number for carriers nowadays typically will be on the order of millions or even billions. Thus, both the full enumeration of pairings and explicitly solving the corresponding formulation will be computationally intractable. The delayed column generation framework has been widely used to address this issue, where pairings are generated on demand driven by the dual values of the LP-relaxed original formulation. The core of this approach is its pricing problem formulation and the corresponding solution methods. Lavoie et al. (1988) constructed an expanded duty-based network such that all paths are in a one-to-one correspondence with the valid crew pairings. Barnhart et al. (1994) proposed a time-line network as a variant of the commonly-used time-space network to reduce the size. Yao et al. (2005) built a pricing problem based on a duty-based network for each crew respectively. Multiple pairings with negative reduced costs were identified among the shortest paths from each of these networks through the Dijkstra's algorithm.

When more complicated requirements on pairings are taken into consideration, paths in the network may not have a one-to-one correspondence to valid crew pairings, and thus finding the shortest path is no longer sufficient to solving the pricing problem. In this case, a shortest path problem with resource constraints is typically modeled in the literature to enforce the requirements that cannot be ensured by the network structure (Dunbar et al., 2012; Shao et al., 2015; Ruther et al., 2016; Cacchiani and Salazar-González, 2017; Wei and Vaze, 2018). This model can be solved exactly by a general labeling algorithm, but in theory, such a process will take exponential time in the worst case (Warburton, 1987).

Therefore, different strategies to speed up the labeling process or directly accelerate the whole column generation process have been developed. Shao et al. (2015) proposed a perturbed Lagrangian dual approach along with a specialized deflected sub-gradient optimization scheme to avoid stalling of the column generation process. Pre-processing to prune arcs and nodes from the network, derive desirable paths up front, and help prune sub-paths during the labeling process more efficiently was applied by Ruther et al. (2016). Wei and Vaze (2018) solved the pricing problem via a two-phase strategy. Before applying the labeling algorithm to the original model, a relaxed version, where paths from different crew bases were allowed to dominate each other, was first solved to check whether desirable pairings could just be identified.

Rather than solving the pricing problem as a shortest path problem with resource constraints, Anbil et al. (1998) performed a depth-first traversal of the underlying duty-based network. During this process, the feasibility of the path was always ensured, while a tally of reduced costs (based on different cost structures) were kept track of.

To obtain a tighter LP-relaxation bound, Vance et al. (1997) introduced an additional set of binary decision variables to the conventional set partitioning formulation, each corresponding to a set of duties that partition the scheduled flights in the planning horizon. This way, the pairing problem can be viewed as split into two phases, where they first decide a way to partition the flights into a set of disjoint duties, and then construct crew pairings just based on this duty set.

Since the delayed column generation approach converges to an optimal solution to the LP-relaxation of the original set partitioning/covering formulation, further mechanisms are required to handle the integrality constraints. Lavoie et al. (1988); Barnhart et al. (1994) and Dunbar et al. (2012) determined an integer solution by directly solving the original, integrality-constrained formulation based on the pairings generated during the column generation process. On the other hand, to ensure the original formulation is solved to IP optimality, the branch-and-price framework can be applied (Desaulniers et al., 1997; Freling et al., 2004; Yao et al., 2005). However, this exact approach could be computationally expensive and time consuming, so several branching strategies and heuristics have been proposed in the literature. Anbil et al. (1998) and Ruther et al. (2016) proposed a dive-and-price method, where follow-ons were only fixed to 1 for each branching. This method was also adopted by Wei and Vaze (2018) but the branching was directly on the decision variables (i.e., pairings) instead of follow-ons. Shao et al. (2015) performed the exact branch-and-bound process on the follow-ons, until a sufficiently large number of columns have been generated while no integer solution has been found, at which point they too solve the associated restricted master problem instead.

Additionally, alternative approaches that are independent of the delayed column generation framework or even independent of the set partitioning/covering formulation, can be found in the literature as well (Emden-Weinert and Proksch, 1999; Ozdemir and Mohan, 2001; Yan and Tu, 2002; Guo et al., 2006; Deng and Lin, 2011; Haouari et al., 2019).

In order to improve the robustness of the schedule and achieve additional overall savings of the operational cost for airlines, some work in the literature proposed to integrate the crew pairing/scheduling problem with several other steps in the overall planning procedure and address them simultaneously rather than in a sequential manner. For example, Cohn and Barnhart (2003); Weide et al. (2010); Dunbar et al. (2012) considered the aircraft routing problem and the crew pairing problem at the same time. In addition, the schedule planning phase was partially incorporated by Klabjan et al. (2002) through allowing the departure time to be moved within a small time window, to grant more flexibility to the crew pairing phase. Sandhu and Klabjan (2007); Shao et al. (2015); Cacchiani and Salazar-González (2017), respectively, proposed integrated models and combined the fleet assignment, aircraft routing, and crew pairing phases together. Ruther et al. (2016) dealt with an integrated aircraft routing, crew paring, and tail assignment problem in a rolling 7-day manner, where fights in the first four days must be covered exactly once by an aircraft and a crew, while only the crew pairing was incorporated for the last three days.

It is worthwhile to mention that there are other problems similar to crew scheduling / pairing problems, in particular similar to our problem studied in this chapter, while out of the scope of aviation operations, for instance, rolling stock scheduling and crew scheduling in railways (Şahin and Yüceoğlu, 2011; Borndörfer et al., 2016; Zhong et al., 2019), and vehicle routing and driver scheduling for long-haul trucks (Rancourt et al., 2013; Rancourt and Paquette, 2014; Koç et al., 2017). Although a similar network representation (e.g., time-space network) is often used, the modeling along with the associated solution approach to these problems is in general not related or applicable to airline crew scheduling / pairing problems because of the differences in the nature of the operations and the associated requirements. For example, rolling stock scheduling needs to model the unit coupling and decoupling (a.k.a. composition changes) between connections, while labor rules in airline crew scheduling have a much more complicated structure than the maintenance and base capacity requirements in rolling stock rotations. Truck routing and driver scheduling are delivery problems, which typically need to determine the time for the trips and other activities, considering the allowable time window(s) for each customer, instead of meeting a predetermined, fixed time point for each trip as in aviation operations. In addition, these truck transportation problems usually need to take additional constraints like vehicle capacity explicitly into account.

2.4 Model and Solution Framework

2.4.1 A Set Packing Formulation

As mentioned above, not all flights scheduled in the planning horizon are required to be covered by line/bid crews. Instead, the cargo airline wants to minimize the cost of using reserve crews for uncovered flights. Therefore, rather than the commonly-used set partitioning or set covering problem, as typically seen in the passenger aviation literature, we treat this problem as a set packing problem with the following formulation:

$$(I_{SP}) \qquad \begin{aligned} \min_{x_p} \quad & -\sum_{p\in P}\sum_{f\in p} c_f \cdot x_p \\ \text{s.t.} \quad & \sum_{p\in P} a_{f,p} \cdot x_p \leq 1 \qquad \forall f \in F \\ & x_p \in \{0,1\} \qquad \forall p \in P. \end{aligned}$$

Here, P denotes the set of feasible crew pairings respecting all of the seven requirements described in Section 2.1 (i.e., leg consistency, labor regulations and worker satisfaction, and break restrictions), F corresponds to the set of scheduled flights in the planning horizon, while whether the pairing contains a specific flight f or not is indicated by the boolean parameter $a_{f,p}$. For notation simplicity, we also indicate a flight f being included in a specific pairing p, i.e., $a_{f,p} = 1$, through $f \in p$. The boolean decision variable x_p specifies whether the corresponding crew pairing p will be assigned to a line crew to operate or not. The constraints ensure that no flight is operated by more than one crew, since carrying extra crews on board is not allowed. The pairing cost c_p is defined by $c_p := -\sum_{f\in p} c_f$, where c_f estimates the excessive cost for operating each flight $f \in F$, if it is not covered by a line crew. Therefore, minimizing $-\sum_{f\in p} c_f$ across all pairings is equivalent to minimizing the total reserve crew cost.

2.4.2 Delayed Column Generation

Initially, we tried a depth-first search (DFS) approach, similar to the ones presented in the literature, to fully enumerate all feasible crew pairings and construct the complete set P above. This resulted in approximately half a million pairings for a 600-flight instance. This huge number indicates that this column generation approach is not appropriate for our problem, as it will take a great amount of time, consume a large amount of storage space (we have to write each generated pairing to the hard disk rather storing all of them in RAM) and cause problems for solving the correspondingly large set packing formulation (I_{SP}).

Instead, we have chosen to implement a delayed column generation framework (as in plenty of crew scheduling work) to handle the very large number of feasible pairings. More specifically, we first consider solving the LP-relaxation of the original set packing formulation:

$$(L_{SP}) \qquad \begin{aligned} \min_{x_p} \quad & -\sum_{p\in P}\sum_{f\in p} c_f \cdot x_p \\ \text{s.t.} \quad & \sum_{p\in P} a_{f,p}\cdot x_p \;\le\; 1 \qquad \forall\, f \in F \\ & x_p \;\ge\; 0 \qquad \forall\, p \in P. \end{aligned}$$

[Note that $x_p \le 1$ is implied by the set packing constraints.]

Rather than solving the above formulation explicitly with the complete set P, we incorporate the crew pairings iteratively on demand, driven by the dual values. Each time we solve formulation (L_{SP}) defined by only the subset of columns (i.e., crew pairings) that have been generated so far (initially, just with slack variables; $P = \emptyset$), i.e., the master problem. After retrieving the dual values π of the corresponding optimal basis, we then seek a new pairing(s) p' whose reduced cost, $c_{p'} - \sum_{f\in F}\pi_f \cdot a_{f,p'} = -\sum_{f\in F}(c_f + \pi_f)\cdot a_{f,p'}$, is strictly less than 0, i.e., the pricing problem, and introduce it to the master problem and thus update the formulation ($P = P \bigcup \{p'\}$). We repeat this master-pricing iteration until no negative reduced cost crew pairing can be identified, which means we have the optimal solution for (L_{SP}).

Once we have solved the LP-relaxation formulation (L_{SP}) to optimality, we then find a feasible solution to the crew pairing problem through a heuristic approach — instead of using a branch-and-price method — by directly applying the mixed integer programming technique to the original integrality-constrained set packing formulation with the set P limited to those generated (columns) pairings, as proposed in Barnhart et al. (1994). This approach typically takes significantly less time compared with the exact branch-and-price approach that branches on follow-up flights (Desaulniers et al., 1997) since we only apply the delayed column generation framework to generate crew pairings at the root node. In addition, we will show in Section 2.6 that this proposed heuristic works effectively, as only a small gap between the heuristic and optimal objective values has been observed.

The key challenge in this delayed column generation framework is how to formulate and solve its pricing problem, that is, how to determine whether there is a negative-reduced-cost pairing or not given the current dual values π and how to generate such crew pairings if one exists. For the rest of this chapter, we will mainly focus on this core part of the delayed column generation framework.

2.5 The Pricing Problem Formulation and Solution Approach

2.5.1 Flight-based Network

To begin, we first construct a flight-based network for modeling the pricing problem of the proposed delayed column generation framework. In this directed network graph $G(V,A)$, each flight in the planning horizon corresponds to a single node. For simplicity, we just use the flight set F to denote the set of corresponding flight nodes in the graph G. An arc from flight f_i to f_j exists if and only if the origin of f_j is the same as the destination of f_i and the gap between these two flights is greater than or equal to the minimum time period required for transition (i.e., the first requirement in Section 2.2 is satisfied). In addition, a dummy source node s and a dummy sink node t are introduced. The source node s points to every flight node $f \in F$ in the graph, while the sink node t is pointed to by each of the flight nodes. That is, $V := F \bigcup \{s,t\}$ and $(s,f),(f,t) \in A$ for $\forall f \in F$. As we can see, each $s-t$ path in this network G corresponds to a potential crew pairing, which will be feasible if all of the six remaining requirements (2–7) listed in Section 2.2 are satisfied. Figure 2.1 provides a simple illustration of this proposed network.

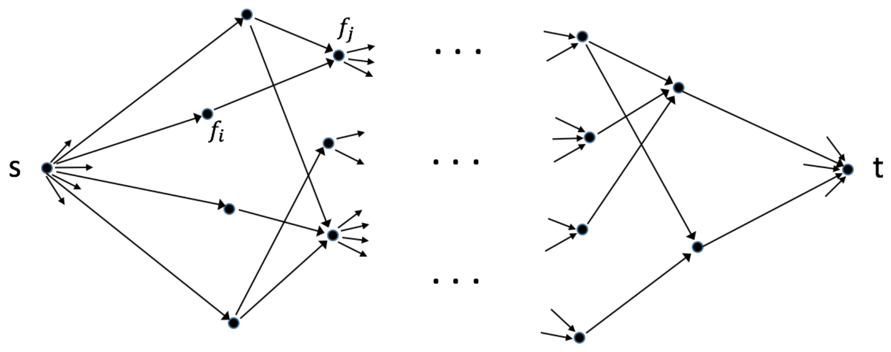

Figure 2.1: The flight-based network.

In the literature, time-space or time-line networks are widely used for crew scheduling and aircraft routing problems, e.g., (Barnhart et al., 1994; Klabjan et al., 2002; Yan and Tu, 2002; Guo et al., 2006; Sandhu and Klabjan, 2007; Liang and Chaovalitwongse, 2013). However, such a structure is not beneficial to represent our cargo flight network for modeling the pricing problem. This is because many airports in our cargo flight network are only associated with a very small number of flights, and therefore, using a time-space or time-line network does not significantly reduce the arcs in the graph representation, while the number of nodes may significantly increase. Furthermore, regarding the crew pairing problem, many networks proposed in the literature are constructed based on the entire set of feasible duty periods, typically generated through an extra enumeration procedure, e.g., (Lavoie et al., 1988; Barnhart et al., 1994; Vance et al., 1997; Yan and Tu, 2002; Yao et al., 2005; Sandhu and Klabjan, 2007). However, as shown in the computational results in Section 2.6 later, the majority of flights in our cargo crew pairing problem are long-haul flights, and therefore, we do not expect a significant difference between the number of viable duty periods and the number of flights, as well as the difference between the associated arcs consequently in the respective networks. In summary, the size of this proposed flight-based network will not differ a lot compared with the traditional time-space/time-line network or a duty-based variation. More importantly, we do not expect a significant difference between these networks in terms of the performance of our proposed solution approaches introduced in later sections. Therefore, we choose to use this flight-based network to formulate the pricing problem, since it's more straightforward to interpret while extra computations (e.g., enumeration, grouping, and sorting) can be avoided.

2.5.2 Shortest Path Problem with Resource Constraints (SPPRC)

To identify negative reduced cost columns (or to ensure that no such columns exist and therefore the current solution to (L_{SP}) is optimal), we propose a pricing problem based on the shortest path problem with resource constraints (SPPRC).

SPPRC was first introduced for solving a routing problem with time windows for bus transportation (Desrosiers et al., 1984). This framework has since been generalized, and several variants have been proposed, to address a wide range of routing and scheduling problems in transportation (Desrochers and Soumis, 1989; Dumas et al., 1991; Feillet et al., 2004), including the airline crew scheduling problems (Vance et al., 1997; Gamache et al., 1998; Kasirzadeh et al., 2017; Wei and Vaze, 2018). In this section, we propose an SPPRC based on the flight-based network described in Section 2.5.1 to solve the pricing problem of the proposed delayed column generation framework. We follow the concepts and terminologies introduced in Irnich and Desaulniers (2005) in the formulation that follows.

We define in total seven constrained resources, denoted as $r_1, r_2, \ldots, r_7$, for modeling our SPPRC. That is, each specific path $p = (s, p_1, p_2, ..., p_k)$ in the network graph G is associated with a **resource vector** $T^p := (T_1^p, T_2^p, \ldots, T_7^p) \in \mathbb{R}^7$ to represent the resource consumption accumulated along this path from the source node s to node p_k, where each entry of this resource vector T_i^p presents the status of resource r_i for the corresponding (sub-)pairing $(p_1, p_2, \ldots, p_k)$ upon the completion of flight p_k.

More specifically, each of the first six resources corresponds to one of the requirements (i.e., bullets 2–7) in Section 2.2, while the last resource is for the reduced cost calculation:

r_1: The amount of time the current duty period has spanned so far (including both flight time and sit-time between flights).

This resource is defined to enforce Requirement (2) in Section 2.2 that no duty period exceeds the upper bound on duty length (denoted as d_{max}).

r_2: The cumulative amount of time the crew has flown so far during the current duty period.

This resource is defined to prevent the violation of the maximum flight time within each duty period (denoted as f_{max}), which corresponds to Requirement (3).

r_3: The number of duty periods the crew has already completed (plus the current one) since their last day-off.

To enforce Requirement (4), this resource value must never exceed the given upper bound on the number of duty periods (denoted as c_d) the crew can consecutively operate without a day off (i.e., a rest period of at least the given minimum amount of time o_{min}, which is longer than the minimum requirement r_{min} for just having a layover) in between.

r_4: The amount of time the current crew pairing has spanned so far (including all duty time, layover time, and day-off time).

This resource is defined to track whether the maximum length of the crew pairing (denoted as p_{max}), i.e., Requirement (5), is violated or not.

r_5: The remaining amount of total time required by the current crew pairing to fulfill the minimum requirement on the whole time span.

This resource corresponds to Requirement (6) in Section 2.2. It is counted down from the minimum target on the length of the whole pairing (denoted as p_{min}) as the crew sequentially completes each of the assigned flights in the pairing.

r_6: The remaining amount of flight time required by the current crew pairing to fulfill the minimum requirement on the cumulative flight time.

This resource is defined to enforce Requirement (7) that the cumulative flight time of the pairing must be at least the given lower bound (denoted as l_{min}), which works similarly to resource r_5 above.

r_7: The reduced cost of the current crew pairing.

This is, the negative value of the summation of the excessive costs for using reserve crews to operate the flights contained in the pairing and also the dual values of the corresponding constraints in (L_{SP}).

Each arc $(i,j) \in A$ in the network G is associated with a **resource extension function (REF)**, $h_{i,j} : \mathbb{R}^7 \to \mathbb{R}^7$, which is used to update the resource vector when moving from node i to j. That is, suppose $p = (s,\ldots,i)$ is a path in the network whose resource consumption is specified by its resource vector T^p, and suppose we extend p via arc (i,j) to get a new path $\bar{p} = (s,\ldots,i,j)$. Then,

$$(ER) \qquad T^{\bar{p}} = h_{i,j}(T^p).$$

The definition of the REFs can naturally be derived, since the way the resource consumption is updated when additional flights are appended to the current pairing should be consistent with how each corresponding resource is defined above. More specifically, these REFs do resource-wise consumption augmentation (for r_4 and r_7), subtraction (for r_5 and r_6), or augmentation with reset (for r_1, r_2 and r_3, depending on the length of the idle period between the corresponding two flights). The complete, detailed definition of $h_{i,j}$ is categorized as follows:

A) Initializing the Path with the First Flight: arcs pointing from the source node, i.e., for $\forall (s,f) \in A$:

$$h_{s,f}(T) = (trv_f,\ trv_f,\ 1,\ trv_f,\ p_{min} - trv_f,\ l_{min} - trv_f,\ -c_f - \pi_f).$$

This is a boundary scenario, where we initialize the calculation of the resource consumption (regardless of the resource vector the dummy initial path $p := (s)$ carries). We initialize the span and the cumulative flight time of the current duty period (i.e., r_1 & r_2) with trv_f, i.e., the flight time of f, since f will be the first flight in the first duty period of this potential crew pairing. Similarly, r_3 is set to 1, counting the number of consecutive duty periods before a day off. We also assign trv_f as the current time length of the pairing (r_4). Since we are keeping track of the remaining span and flight time needed for satisfying the corresponding lower bounds, we subtract

trv_f from the target values p_{min} / l_{min} for r_5 / r_6, respectively. Lastly, we update the reduced cost by initializing r_7 with $-c_f - \pi_f$. The meaning and the correctness of this calculation can be derived from the expression of the reduced cost of a specific pairing g, which is $-\sum_{f \in F}(c_f + \pi_f) \cdot a_{f,g}$ based on the notation introduced in Section 2.4.2. As this expression can be written as $\sum_{f \in g}(-c_f - \pi_f)$, we have the reduced cost of a pairing increasing by $-c_f - \pi_f$ if a new flight f is appended to it.

B) Completing the Path: arcs pointing to the sink node, i.e., for $\forall (f,t) \in A$:

$$h_{f,t}(T) \ = \ T.$$

This is the other boundary scenario, where we obtain an $s-t$ path and thus the corresponding crew pairing becomes complete. Since t is just a dummy node representing the termination of the pairing, no changes to the resource vector T are warranted.

C) Augmenting the Path with the Next Flight: arcs pointing from one flight node to another flight node, i.e., for $\forall (i,j) \in A$, where $i,j \in F$
(denote the departure time and arrival time of a specific flight i as dpt_i and arr_i, respectively, and denote the summation of the connection time between two flights i and j plus the duration of the second flight j itself as $aug_{i,j}$; that is, $aug_{i,j} := arr_j - arr_i$.):

c1) If the idle time between flight i and j is not long enough for a layover (i.e. $dpt_j - arr_i \ < \ r_{min}$) and thus flight i and j will be in the same duty period:

$$h_{i,j}(T) \ = \ T \ + \ (aug_{i,j},\ trv_j,\ 0,\ aug_{i,j},\ -aug_{i,j},\ -trv_j,\ -c_j - \pi_j).$$

Since the gap between the two adjacent flights i and j is less than r_{min}, the span of the current duty period (i.e., r_1) should be increased by $aug_{i,j}$ and the cumulative flight time within the current duty period (i.e., r_2) should be increased by trv_j, while the count of the total duty periods since the crew's last day off (i.e., r_3) should remain unchanged, because in this scenario flight j is still in the same duty period as flight i. A value of $aug_{i,j}$ will be augmented to the consumption of resource r_4, as the time span of the corresponding pairing will grow by this amount. By the same logic, we update the consumption of r_5 and r_6, but rather than augmenting the corresponding amount we need to respectively do subtraction according to the definition of these two resources, which are used to enforce the minimum requirements on the length and cumulative flight time of the pairing, as presented previously. Lastly, $-c_j - \pi_j$ is added to r_7 for calculating the reduced cost of this extended pairing (with the new flight j), based on the analysis provided in scenario A) above.

c2) If flight j will be in a new duty period rather than in flight i's, but the layover between these two flights is not long enough for the crew to have a day off (i.e., $r_{min} \leq dpt_j - arr_i < o_{min}$):

$$h_{i,j}(T) = (trv_j,\ trv_j,\ T_3+1,\ T_4+aug_{i,j},\ T_5-aug_{i,j},\ T_6-trv_j,\ T_7-c_j-\pi_j).$$

In this scenario, the consumption of resources r_4, r_5, r_6, and r_7 is updated in exactly the same way as in the above scenario c1), for exactly the same reason. With respect to resources r_1, r_2, and r_3, their calculation becomes different, because now the crew will have a layover between the two flights. The "clock" for r_1 and r_2 are both reset to trv_j, since a new duty period starts at flight j after the layover. The consumption of r_3 is increased by one unit because the corresponding rest period is not long enough to for the crew to have a day off.

c3) Otherwise, the crew will have a day off between flight i and j (i.e., $dpt_j - arr_i \geq o_{min}$):

$$h_{i,j}(T) = (trv_j,\ trv_j,\ 1,\ T_4 + aug_{i,j},\ T_5 - aug_{i,j},\ T_6 - trv_j,\ T_7 - c_j - \pi_j).$$

The consumption of all resources but r_3 is updated in exactly the same way as in the previous scenario c2), since in this scenario the crew will indeed have a layover and start a new duty period after that with flight j as the first flight. However, as the layover is long enough to be a day off, the counter for r_3 will be rolled back to 1, indicating the crew is working on the first duty period after the day off.

In order to ensure the feasibility of the corresponding (sub-)pairing of a specific path, each node $v \in V \backslash \{s\}$ in the network G is assigned a static **upper bound**, $U^v \in \mathbb{R}^7$, for restricting the resource consumption on the path:

$$\begin{aligned} U^f &= (d_{max},\ f_{max},\ c_d,\ p_{max},\ \infty,\ \infty,\ \infty) && \forall f \in F \\ U^t &= (d_{max},\ f_{max},\ c_d,\ p_{max},\ 0,\ 0,\ \infty) && \text{for the sink node } t. \end{aligned}$$

We say that a specific path $p = (s, p_1, p_2, \ldots, p_k)$ is feasible with respect to all resource constraints if and only if the resource consumption specified by the resource vector of this path p as well as all of its sub-paths are always within the corresponding upper bound, i.e., if and only if resource vector $T^{p^i} \leq U^{p_i}$, for $i = 1, 2, \ldots, k$, where $p^i = (s, p_1, p_2, \ldots, p_i)$ is the sub-paths of p.

The definition of the upper bound U^v on resources r_1, r_2, r_3, r_4, and r_7 for all $v \in V \backslash \{s\}$ is straightforward, given the way these resources are defined at the beginning of this sub-section. For resources r_5 and r_6, there is no consumption limitation defined for the intermediate paths during the middle of the extension (i.e., $U_5^f = U_6^f = \infty$ for $\forall f \in F$), since each of them may still have the potential to be extended to a complete $s-t$ path which achieves the minimum requirements on the length and the cumulative flight time (i.e., Requirement (6) & (7)) with respect to its corresponding crew pairing later. However, when we reach the sink node t, i.e., when the crew pairing is fixed to be complete, the zeros in the fifth and sixth coordinate of U^t require the values on both r_5 and r_6 to be non-positive, which ensures that these two minimum targets have indeed been achieved already. Through this definition, we can easily verify that the feasibility of a crew pairing is equivalent to the feasibility of its corresponding $s-t$ path in the network G with respect to all resource constraints.

Note that if at some point the resource vector of a specific path on r_5 / r_6 is non-positive, we know that the minimum requirement on the length / flight time of the pairing should have already been satisfied. Since additional flights are being appended to the pairing as we extend the path, neither of these two requirements will be violated once satisfied. Therefore, in the actual implementation of this SPPRC, we also introduce a lower bound to each node in the network to do round-ups (to zeros) for these two resources, once the value of resource vector on them becomes non-positive. It is computationally preferable to avoid the value (the remaining amount of time) for resources r_5 and r_6 being negative because otherwise the labeling process for solving the pricing problem will be slowed down (see Algorithm 1 in Section 2.5.3 for details). Consequently, this lower bound will work together with the upper bound, and thus form a resource window for each node, which follows the more general SPPRC formulation introduced in Irnich and Desaulniers (2005). We refer the reader to this paper for more detail, while we instead present the more straightforward implementation here for clarity of exposition.

Let $\mathscr{P}$ be the set of all feasible $s-t$ paths with respect to all resource constraints in the network G. Then, the original pricing problem, i.e., determining whether there is a negative-reduced-cost pairing and generating such a crew pairing if there exists one, is equivalent to solving the following formulation based on our SPPRC model:

$$(PP) \qquad \min_{p \in \mathscr{P}} T_7^p.$$

(That is, we look for the minimum value of r_7 among the resource vectors of all feasible $s-t$ paths in the network.)

More specifically, if the optimal objective value of (PP), denoted as z, is strictly less than 0, then a path p that achieves this z corresponds to a negative reduced-cost pairing. On the other hand, if $z \geq 0$, that means no negative-reduced-cost crew pairing exists under the provided dual values.

2.5.3 Labeling Algorithm

Before we present the algorithm for solving formulation (PP), we first describe two properties of our SPPRC model that play an important role in the algorithm implementation.

(a) The flight-based network $G(V,A)$ is a directed, acyclic graph. Therefore, we can topologically order all nodes in the network graph G by applying Kahn's algorithm (Kahn, 1962).

(b) According to the REFs defined in Section 2.5.2, there exist no interdependencies between different resources during the extension. In addition, for each resource, the corresponding REF on this resource for any arc in the network is always an affine function with non-negative linear coefficients. That is, denoting the REF for arc (i,j) on resource r as $h^r_{i,j}(\cdot)$, we have $h^r_{i,j}(T) = \alpha^r_{i,j} T_r + \beta^r_{i,j}$, where $\alpha^r_{i,j}, \beta^r_{i,j} \in \mathbb{R}$ and $\alpha^r_{i,j} \geq 0$ for all arcs (i,j) and resources r.

Particularly based on property (b) above, we can derive the following proposition for our model:

Proposition 2.1. Given two specific paths $\bar{p}$ and $\hat{p}$ that have the same *resident node* (i.e., the last node of the path), if the resource vector of $\bar{p}$ *dominates* the resource vector of $\hat{p}$ (i.e., $T^{\bar{p}} \leq T^{\hat{p}}$ but $T^{\bar{p}} \neq T^{\hat{p}}$), then for any extension path e from that resident node, we have $T^{(\bar{p},e)} \leq T^{(\hat{p},e)}$.

Proof. Denote the resident node of $\bar{p}$ and $\hat{p}$ as u. To prove this proposition, we only need to show that for any $(u,v) \in A$, path $\bar{p}' = (\bar{p},v)$ has a resource vector dominating or exactly the same as that of path $\hat{p}' = (\hat{p},v)$, i.e., $T^{\bar{p}'} \leq T^{\hat{p}'}$, because the original statement can be argued by recursively applying this conclusion. By property (b) presented above, we know that $T_r^{\bar{p}'} = h_{u,v}^r(T^{\bar{p}}) = \alpha_{u,v}^r T_r^{\bar{p}} + \beta_{u,v}^r \leq \alpha_{u,v}^r T_r^{\hat{p}} + \beta_{u,v}^r = h_{u,v}^r(T^{\hat{p}}) = T_r^{\hat{p}'}$ for arbitrary resource r, because $T^{\bar{p}} \leq T^{\hat{p}}$ and $\alpha_{u,v}^r \geq 0$. Therefore, we have $T^{\bar{p}'} \leq T^{\hat{p}'}$. □

By this proposition, if the resource vector of a feasible path $\hat{p}$ is dominated by, or equal to another feasible path $\bar{p}$'s (i.e., if $T^{\bar{p}} \leq T^{\hat{p}}$), then we can discard $\hat{p}$ without considering its extensions, because $\bar{p}$ can always provide a no worse alternative.

Based on this proposition plus the fact that all nodes can be topologically ordered, we develop a labeling algorithm (see Algorithm 1) to solve formulation (*PP*), which thus solves the pricing problem.

In sum, we extend the dummy path $p = (s)$ along the network towards the sink node t to generate $s-t$ paths. However, instead of traversing all sub-paths in the network during this process, we prune out those infeasible and inferior ones, based on the properties introduced above. Among all $s-t$ paths we get this way, we dynamically keep track of the one whose corresponding crew pairing has the smallest reduced cost, and thus eventually return an optimal solution to the pricing problem formulation (*PP*).

Algorithm 1 Labeling

```
1:  Topologically order all nodes in the network using Kahn's algorithm (Kahn, 1962);
2:  Initialization: intermediate sub-path container $\mathscr{U} = \{(s)\}$, $result$ = NULL, and $obj = +\infty$;
3:  while $\mathscr{U} \neq \emptyset$ do
4:      Choose a path $p \in \mathscr{U}$ whose resident vertex, denoted as $v(p)$, has the smallest topological order;
        Remove $p$ from $\mathscr{U}$;
5:      for all possible one-step extensions for $p$, i.e., for all arcs $(v(p),n) \in A$ do
6:          Calculate the resource vector $T^{\bar{p}}$ for the the extended path $\bar{p} := (p,n)$;
7:          if $T^{\bar{p}} \leq U^n$, i.e., path $\bar{p}$ is feasible with respect to all resource constraints then
8:              if $\bar{p}$ is an $s-t$ path (i.e., $n == t$), and achieves a smaller reduced cost than $obj$ then
9:                  Update the output variables accordingly: $result = \bar{p}$ and $obj = T_7^{\bar{p}}$;
10:             else if $\bar{p}$ is not an $s-t$ path then
11:                 Loop through all paths $\hat{p} \in \mathscr{U}$ with $v(\hat{p}) == n$, and remove those whose resource vector
                    is dominated by or equal to $\bar{p}$'s (i.e., $T^{\bar{p}} \leq T^{\hat{p}}$) from $\mathscr{U}$;
12:                 if during this process, no path $\hat{p}$ is found dominating $\bar{p}$ then
13:                     Add $\bar{p}$ to $\mathscr{U}$;
14: return $result$ and $obj$
```

2.5.4 Speed-up Strategies and Improvements

A simple application of the labeling algorithm presented in the previous section may not be able to effectively solve the LP-relaxation of the set packing model (L_{SP}) in practice, considering the size of the problem and the complexity of the requirements. For instance, we tested the proposed approach on a real-world instance, where there are in total 606 flights scheduled across a monthly planning horizon. Our experiment required 929 iterations between the master and pricing problem with the runtime of each iteration ranging from 15 to 70 seconds, in total requiring about 12.5 hours to converge. Such computational performance may delay the schedule implementation and deployment; thus we must speed up this solution process in order to make our proposed method tractable for practical use.

We propose the following to reduce runtime:

(I) During the labeling process, we prune out "short" paths which cannot possibly be extended to a full pairing that satisfies the minimum requirements. To achieve this, we apply backward dynamic programming up front, to calculate, for each flight vertex v, the *maximum possible span* (denoted as $maxSpan^v$) as well as the *maximum* *possible cumulative flight time* (denoted as $maxFly^v$) starting from there to the sink vertex. Based on this, we additionally discard paths $\bar{p}$ during the labeling process that cannot satisfy condition $T_5^{\bar{p}} \leq maxSpan^n$ or $T_6^{\bar{p}} \leq maxFly^n$, where n is denoted as the resident node of $\bar{p}$. This modification is expected to effectively provide a speed-up because the values of r_4 and r_5 in the resource vector for any path will be complementary (i.e., their summation will always equal p_{min}) during the early stage of the extension process, which means we can rarely prune out any feasible paths by just the dominance rule at the beginning.

(II) We enable multiple pairings to be added to the master problem at each iteration. If we identify more than one s-t path during the solution process that corresponds to a pairing with negative reduced cost, we add up to K of them, specifically those with the most negative reduced costs, to the master problem. We need to use such an upper bound K particularly because there could be a big number of desirable pairings identified during the early stage of the column generation, while we cannot afford to bring all of them to the master problem.

(III) This is a variation of the previous strategy. Rather than the most negative pairings, we add the first ones (up to the bound K) we find during the labeling process to the master problem. Given that pairings achieving smaller reduced costs are not necessarily more beneficial, rather than spending time searching for the most negative ones among all $s-t$ paths reached by the algorithm, we stop earlier to reduce runtime spent on solving the pricing problem.

(IV) Before a feasible, non-$s-t$ intermediate path is about to be pruned through dominance, we additionally check whether or not it already corresponds to a feasible, negative reduced cost crew pairing. If so, we treat this pairing as a candidate, which potentially can be added to the master problem (as one of the K most negative or the first K negative reduced cost pairings, depending upon whether we implement Improvement (II) or (III) above).

In Improvement (II) or (III), we seek to derive the K most negative or the first K negative reduced cost pairings from the identified $s-t$ paths, which are all together added to the master problem to help reduce the total number of iterations needed to solve the LP-relaxation (i.e., L_{SP}) to optimality. However, when the total number of such identified $s-t$ paths is relatively small, it may be beneficial to add valid pairings that have negative reduced cost even if their corresponding paths are dominated by some others. Thus, in Improvement (IV), when we prune a dominated path, we also check to see if the two minimum requirements (6) and (7) in Section 2.2 have been satisfied by its corresponding crew pairing and if its reduced cost is negative. If so, we still consider that pairing as a candidate for the K most negative reduced crew pairings to be added to the master problem. In addition, this improvement can also mitigate the tailing effect of the delayed column generation process.

2.6 Computational Experiments

We implemented our proposed model associated with the labeling algorithm as well as the four speed-up improvements using C++ (Visual Studio 2017) with CPLEX (version 12.80) on a 64-bit operating system computer with two 2.10GHz processors and 128GB RAM.

Note that for all the following experiments for all instances, we treat the excessive cost for operating an uncovered flight by a reserve crew equally across all scheduled flights. In other words, we have $c_f = 1$ for all flights $f \in F$ in formulation (I_{SP}) and (L_{SP}), and therefore we are simply maximizing the number of flights covered by crew pairings. We conduct our experiments with this specific objective parametrization because our partnered cargo airline views flight coverage as the highest priority at the moment, and works under the assumption that having less flights that need to be assigned to reserve crews will result in lower overall costs in practice.

Before we present the results of testing the computational performance of the final solution approach on a set of previous practical instances from our partnered cargo airline, we first evaluate the effectiveness of the proposed improvements through solving the 606-flight instance mentioned at the beginning of Section 2.5.4.

For convenience of presentation, we differentiate strategies (III) and (II) + (III) for the rest of the experiments, where we refer to (III) as the version where we simply terminate the labeling process in algorithm 1 once an $s-t$ path that corresponds to a negative reduced cost crew pairing has been identified, while (II) + (III) corresponds to the original statement, i.e., adding the first K negative reduce cost pairings identified, as described in the third bullet of Section 2.5.4.

Our experiments show that the mechanism in (I) helps reduce the solution time for each iteration of column generation from 15 to 70 seconds to 7 to 40 seconds, and in total reducing about 30% runtime for solving the LP-relaxation of the set packing formulation (L_{SP}). Table 2.1 shows that if we just terminate the labeling process once we find an $s-t$ path that corresponds to a negative reduced cost crew pairing, the “first-negative” strategy presented in (III) stand-alone will actually increase the overall runtime because much more iterations will be taken. However, when embedded into the strategy (II) so that multiple pairings are allowed to be added to the master problem each time, both the original “most-negative” and the “first-negative” strategies will perform very well when the parameter K reaches a certain big number, as illustrated in Figure 2.2.

Table 2.1: Computational experiments testing the effectiveness of embedding Improvement (I) and/or (III) into the labeling process. Here, **O** refers to the original approach as presented in Algorithm 1, where a valid crew pairing with the most negative reduced cost will be identified and added to the master problem.

Approach	#Iterations	LP-Runtime
O	929	12hr 37min
O+(I)	896	8hr 13min
O+(III)	5771	2day 5hr
O+(I)+(III)	5899	1day 14hr

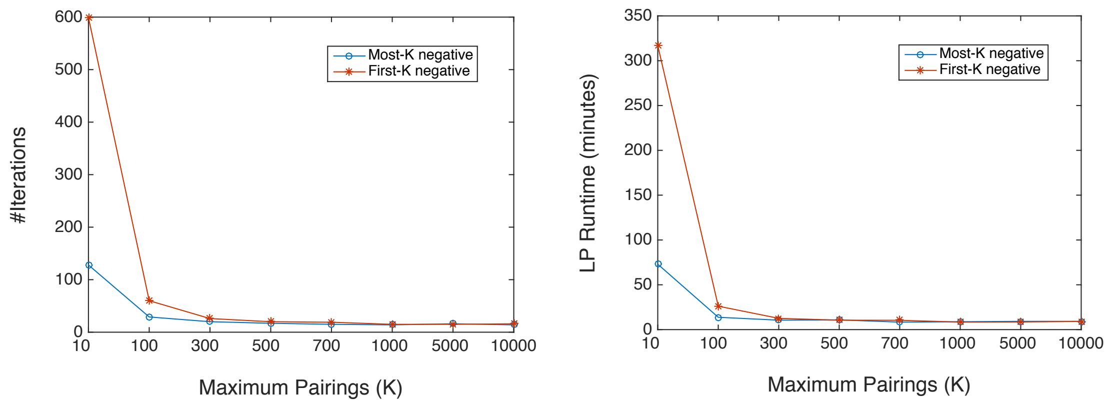

Figure 2.2: The number of iterations and runtime as functions of the value of parameter K for both the "most-negative" strategy and the "first-negative" strategy, i.e., Approach **O+(II)** and Approach **O+(II)+(III)**.

Observe that the more pairings we feed to the master problem per iteration, the faster we can solve the whole problem (L_{SP}), but when K is large, there are not that many $s-t$ paths with a negative reduced cost in the network for the algorithm to identify. This motivated us to consider Improvement (IV) in Section 2.5.4. More specifically, since a specific feasible $s-t$ path will be found during the original labeling process only if none of its sub-paths are dominated by some others, this strategy can "relax" this requirement, and thus make it possible to set a higher but still effective K.

By this additional strategy, together with previous improvements (I) and (II), we can eventually solve the formulation (L_{SP}) for the considered 606-flight instance in a very short time. Table 2.2 below provides a summary of the computational performance of this finalized solution approach, i.e., **O+(I)+(II)+(IV)**, and also a comparison between approaches with different improvements/strategies for demonstrating their effectiveness.

Table 2.2: Computational performance of the proposed labeling algorithm with different improvements and strategies incorporated for solving the real-world 606-flight instance, where $K = 20,000$ if Improvement (II) is applied.

Approach	LP-obj	#Iter.	Runtime	#Pairings Gen.	IP-obj	Runtime
O	336.382	929	12hr 37min	929	321	5min 14sec
O+(III)	336.382	5771	2d 5hr	5,771	331	1min 19sec
O+(I)	336.382	896	8hr 14min	896	316	46min 28sec
O+(I)+(II)	336.382	16	9min 17sec	5,949	330	6min 21sec
O+(I)+(II)+(IV)	336.382	10	6min 38sec	23,492	332	2min 38sec

Here, the first column in Table 2.2 shows the approach we use for solving the pricing problem of the proposed delayed column generation framework. The second column displays the final objective value (actually, the negative of the objective value, as it can then represent the number of flights covered) we get for the LP-relaxation of the set packing formulation (i.e., L_{SP}). The number of iterations taken, the total time spent, and the number of valid crew pairings generated during the delayed column generation process for solving (L_{SP}) are provided in the third, fourth, and fifth column, respectively. The sixth column shows the (negative) objective value we can achieve by solving the restricted set packing formulation (I_{SP}) with just crew pairings generated during the delayed column generation process (i.e., the heuristic approach described in Section 2.4.2), while the corresponding time spent for solving this formulation using the branch-and-cut approach by CPLEX is presented in the last column.

To further verify the effectiveness and robustness of the finalized approach (i.e., the labeling algorithm with Improvement (I), (II), and (IV)), we apply it to solve two additional real-world instances. Table 2.3 summarizes the basic information of each of the three instances (including the previous 606-flight one). Table 2.4 provides some general information on the flight-based network we proposed in Section 2.5.1, and also shows the full enumeration results for each of the three instances, where the entire set of valid crew pairings are generated by a depth-first-search (DFS) approach, and then are used to explicitly solve the original set packing formulation (I_{SP}) to optimality.

Table 2.3: General information for the three real-world instances from our partnered cargo airline. "Long-Haul" shows the percentage of flights which have a flight time greater than 4 hours. "Int-Int" refers to flights whose origin and destination are both outside the U.S., while "Dom-Int" refers to flights flying from some place in the U.S. to somewhere outside the country.

Instance	#Flights	Long-Haul	Int-Int	Int-Dom	Dom-Int	Dom-Dom
No.1	606	92.41%	49.84%	23.93%	25.41%	0.82%
No.2	541	92.24%	50.28%	25.14%	24.03%	0.55%
No.3	644	86.96%	47.36%	23.91%	23.91%	4.81%

Table 2.4: A summary of the underlying flight-based network and the full enumeration results for the three instances.

Instance	#Nodes	#Arcs	#Valid Pairings	Enum. Time	#Flt. Cov.	Soln Time	Coverage
No.1	608	12,539	440,641	30min 34sec	332	1min 14sec	54.79%
No.2	543	10,113	329,145	26min 40sec	281	2min 42sec	51.94%
No.3	646	12,201	462,395	35min 52sec	334	7min 55sec	51.86%

Here, the second and third column in Table 2.4 respectively displays the number of nodes and arcs in the flight-based network. The number of valid crew pairings satisfying all of the seven requirements introduced in Section 2.2 is presented in the fourth column, while the time for enumerating all of these pairings by DFS is shown in the fifth column. The sixth column provides the optimal objective value (in negative) for solving the set packing formulation explicitly, and the corresponding solution time by the branch-and-cut approach by CPLEX is provided in the seventh column. The last column calculates the flight coverage achieved by the optimal solution.

The computational performance of our finalized approach on these three instances is provided in Table 2.5. The columns of this table are structured exactly as those in Table 2.2 above, except there is one additional column appended at the end, which displays the corresponding flight coverage achieved for the corresponding instance by our finalized approach.

According to these experiments, the number of iterations and the total runtime for solv-

Table 2.5: Computational results for applying the finalized solution approach ($K = 20,000$) to the three instances.

Instance	LP-obj	#Iter.	Runtime	#Pairings Gen.	IP-obj	Runtime	Coverage
No.1	336.382	11	6min 13sec	22,052	332	28sec	54.79%
No.2	284.447	9	3min 36sec	16,642	281	24sec	51.94%
No.3	340.327	10	6min 24sec	23,736	333	5min 20sec	51.71%

ing the corresponding LP-relaxation (i.e., L_{SP}) are both kept to a very small value. In addition, these experiments overall demonstrate that our proposed approach is able to solve the whole crew pairing problem much faster than the full enumeration (DFS) method, with only a small portion of feasible crew pairings generated through the proposed delayed column generation framework (which is less than 6% for all of our instances). Lastly and more importantly, the final objective value we get through the heuristic, proposed for handling the integrality constraints, is verified to be exactly the same as, or extremely close to, the true optimal value.

2.7 Conclusion

In this chapter, we considered a crew pairing problem for our partnered cargo airline, where the underlying network primarily consists of long-haul, international flights and lacks repeating patterns. As a result, this crew pairing problem needs to partially accomplish the conventional crew rostering phase to directly generate complete flying task schedules for crews during the planning horizon, and therefore an extra set of unique and complex regulations and agreements must be addressed simultaneously.

To tackle these challenges, we have modeled the crew pairing problem as a set packing problem, and proposed a delayed column generation framework — a common way to handle the exponentially-large number of decision variables in the corresponding formulation. We constructed a flight-based network, and based on that, we formulated the pricing problem as an SPPRC, where the generated crew pairings are guaranteed to be valid, with all of the unique and complex regulations and rules incorporated and modeled in an efficient manner. A conventional labeling algorithm, enhanced by several speed-up strategies, was developed to solve this SPPRC, and thus address the pricing problem.

Computational experiments have shown the effectiveness of the proposed speed-up strategies. Furthermore, the results have demonstrated that our finalized approach is able to solve real-world instances in a short time, while it can always produce a set of crew pairings that corresponds to a theoretical optimal solution or is extremely close to optimality. Consequently, our approach has been verified to significantly outperform the manual construction approach currently used by our partnered cargo airline as well as a full column enumeration approach.

Although our proposed approach was effective in solving the crew pairing problem, Table 2.4 shows that even the theoretical optimal flight coverage for the considered real-world instances is uniformly less than 55%, which may be too low to be directly deployed in practice. In the next chapter, we consider the method the cargo airline implements to deal with this low flight coverage issue, and we further develop new modeling and solution approaches to efficiently incorporate it into the crew pairing construction process.

CHAPTER 3

An Arc Selection Approach for Modeling a Potential Break in Cargo Crew Pairings

3.1 Introduction

In this chapter, we consider an extension of the cargo crew pairing problem we discussed in the previous chapter. We begin with the motivation for conducting this research and a brief background for this extension.

In practice, it is frequently not possible to achieve a desirable flight coverage in crew scheduling for our partnered cargo airline, even by the theoretically best solution, if only crew pairings that satisfy all requirements and rules are considered. This has also been demonstrated by our experiments on real-world instances in Section 2.6 in the previous chapter.

Aside from the strict FAA regulations and bargaining agreements for ensuring crew pairings' legality and satisfaction, the low flight coverage issue is primarily due to the nature of the underlying flight network, since it mostly consists of long-haul flights, and lacks opportunities for quick turns, while including many airports with only a small number of associated flights. Consequently, there are only a very small number of possible flight combinations and a very limited number of valid pairings for covering some specific flights.

Unlike passenger aviation, traditional deadheads (i.e., crew members flying between two places as passengers by scheduled flights or other external, commercial flights) cannot be systematically supported in the pairing construction for line crews to increase connectivity in the underlying flight network and thus improve flight coverage because of a variety of different cost and logistical issues associated with deadheading long-haul, international flights. For instance, there is a flight frequency issue. Unlike in the passenger crew scheduling problem where there are an abundant number of flights between airport hubs and spokes, this is not necessarily the case for long-haul flights. With long-haul and international flights, there exists a distinct possibility that, due to the limited number of flights, a crew may need to wait overnight or even a couple of days, away from base, for the deadhead flight. Since the time for deadheading accordingly factors into duty time and crew pairing span limitations, deadheading flights will thus significantly lower crew efficiency and increase operational expenses, which cannot be accepted by the cargo airline.

Given this, in order to address the low flight coverage issue, the cargo airline chooses to allow crews to fly home commercially and take a break in the middle of their respective flying schedule during the month. Since the airports right before/after the break are no longer required to be the same, this break feature, working as a special deadheading method, relaxes the leg consistency requirement. Therefore, it is expected that, by allowing such a break to take place, the number of valid crew pairings for covering each specific flight can be significantly increased, and thus much greater flight coverage can be eventually achieved.

For the remainder of this chapter, we consider incorporating a model of this new break feature into the cargo crew pairing problem that we studied in the previous chapter, evaluate how it impacts the original model and solution framework, and propose a new approach to address the additional challenges introduced by it. More specifically, the remainder of this chapter is structured as follows: in Section 3.2, we present a statement of this crew pairing problem extension, with a detailed description of the additional rules and metrics introduced by the break feature. In Section 3.3, we summarize the previous work in the literature that is related to the break feature considered here, as well as compare it to the approach we propose for effectively modeling it. Section 3.4 introduces a straightforward modification of the model and the associated solution approach proposed in Chapter 2 to solve this extension, and also describes the inadequate computational performance of this modification to solve practical instances. More specifically, we still propose to use a delayed column generation framework, and model its pricing problem as a SPPRC. However, since lots of additional arcs need to be included in the network to incorporate the potential break while additional resources in the SPPRC need to be defined to enforce the associated requirements, the pricing problem cannot be solved directly by the original labeling approach within an acceptable time. Therefore, in Section 3.5, we propose an exact arc selection approach for more efficiently solving this pricing problem of the delayed column generation framework, to address the computational challenges. Section 3.6 presents computational experiment results on real-world instances to evaluate our new approach. Lastly, in Section 3.7, we conclude and provide thoughts for future work.

3.2 Problem Statement

As described previously, to have a break, the crew is allowed to fly commercially home and also to resume the rest of assigned flying tasks during the planning horizon. Therefore, the original leg consistency rule (bullet (1) in Section 2.2) on crew pairing validity is relaxed as the following:

(1*) The origin of a flight should be the same as the destination of its previous flight, with a minimum time period (e.g., 45 minutes) in between for transition, **except** for the flight right after the break (if it exists), whose origin is not required to match the destination of the flight right prior to the break.

Besides the FAA regulations and the airline's collective bargaining agreement (i.e., bullets (2) – (7) in Section 2.2), the following four additional requirements must be satisfied, to incorporate the described break feature into the crew pairing construction:

(8) The crew can have at most one break within the pairing.

(9) The duration of the break (if it exists) should be greater than or equal to a specific lower bound (e.g., 6 days). Note that the time for flying home and back is included in the duration of the break. In other words, the break starts as soon as the final pre-break cargo flight is completed.

(10) The crew cannot have the break before completing a minimum number of flights (e.g., 4 flights).

(11) The crew cannot have the break if a maximum number of flights (e.g., 6 flights) that are already completed is exceeded.

Note that the break, if it exists, counts as a day-off, given its sufficiently long duration, but its duration is not counted into working time, and not counted into the span of the whole pairing either. In other words, this break feature indeed yields a new pairing definition. That is, crew pairings can consist of two segments of flying tasks with a sufficient gap in between (i.e., bullet (9) above), where each segment should respect all local requirements (i.e., bullet (1) – (4) in Section 2.2, and additionally (10) & (11) above for the first segment), while they two collectively meet the rest of global ones that are imposed on the whole pairing (i.e., bullet (5) – (7) in Section 2.2). The last two requirements above, (10) and (11), together ensure that if the crew will take a "break" within the pairing, it should never occur somewhere too early, nor too late, and thus the two flying task segments are balanced.

From the perspective of both a pilot's quality of life and the cargo airline's expenses, having a break in the middle of the pairing is less preferable than the no-break pairing. In order to implement as few breaks as possible, in addition to minimizing the previous excessive cost of flying uncovered flights by reserve crews, a penalty cost is therefore introduced for each crew pairing that has a break, and should be minimized as well.

3.3 Related Work

The maintenance requirements in aircraft routing (Derigs and Friederichs, 2013; Liang and Chaovalitwongse, 2013; Cacchiani and Salazar-González, 2017) and in rolling stock scheduling (Borndörfer et al., 2016; Zhong et al., 2019), and the day-off, weekly/monthly rest period requirements in railways crew scheduling and airline crew rostering (Gamache et al., 1999; Şahin and Yüceoğlu, 2011; Kasirzadeh et al., 2017) are features commonly found in the literature, that separate the corresponding schedule into multiple task segments. According to the description above, the break feature considered in our extended crew pairing problem serves a similar function. However, incorporating this feature in the crew pairing differs greatly from handling those requirements because having a break in the middle of a crew pairing is not mandatory, but only an optional, and undesirable, relaxation the cargo airline accepts in the crew schedule construction to achieve a better flight coverage. In fact, the day-off rule already considered in our crew pairing problem (i.e., bullet (4) in Section 2.2) is much closer to many of those mentioned features in the literature, as they are required to occur periodically, but this new break feature is not subject to such a restriction. In addition, unlike the requirements on aircraft or rolling stock maintenance, which require it to take place at some predetermined spots, and during a specific time of day (e.g., overnight), the assignment of the break in our problem has much more flexibility. Due to these differences, many novel approaches for modeling those widely studied features/requirements are not applicable to our break feature, including the network layering techniques (Şahin and Yüceoğlu, 2011; Liang and Chaovalitwongse, 2013), the hyperarc network representation (Borndörfer et al., 2016), and so on. Instead, to entirely capture all possible breaks across all pairings, we have to explicitly represent them as arcs in our network and formulate the corresponding requirements as extra resources in the SPPRC, which results in an extremely dense network and consequently an unacceptably slow performance of our labeling algorithm (see the next section for more details).

The approach we propose in this chapter to address the computational challenges is based on a dynamic arc assessment and filtering process. This idea is inspired by the work of Barnhart et al. (1995), where they proposed an approach for selecting deadheads to be incorporated into a pairing generation approach. However, unlike their approach, which is only a heuristic, our approach, as its extension, can guarantee the exactness of the selection and ensure the delayed column generation process converges to optimality of the LP.

Our proposed approach is also similar to the bidirectional search approach proposed by Irnich et al. (2010), which is used to accelerate the branch-and-price process through efficiently pruning the branching tree. More specifically, they eliminate arcs in the network permanently using the reduced cost fixing technique, where scanning through the underlying intact network at that moment in both a forward and backward direction is a necessary step in order to calculate the reduced cost (and/or its attainable upper bound) of the corresponding implicitly-formulated arc variables. However, their approach is not applicable to our problem, since we cannot afford to do a full scan due to the high density of the network. Instead, our proposed approach actually targets to address this challenge. Moreover, in our approach, we only temporarily remove a majority of arcs from the network for each iteration of the delayed column generation process, and then determine (a small subset of) those beneficial arcs to be brought back to the network by performing a bidirectional scanning on the remaining sub-network, while no arcs will be deleted permanently. Lastly, as the nature of the reduced cost fixing, the method in Irnich et al. (2010) only provides a sufficient condition for path elimination, but our approach will prune arcs in an exact manner, which is described and demonstrated in more detail in the following sections.

3.4 Model Modification

We can easily modify our approach in the previous chapter to still solve this crew pairing extension with the break feature incorporated. First, we update the objective function in the set packing formulation to reflect the additional penalty cost introduced by the deployment of crew pairings which have a break:

$$(I_{SP}*) \qquad \begin{aligned} \min_{x_p} \quad & \sum_{p \in P} (I_p \cdot c_b - \sum_{f \in p} c_f) \cdot x_p \\ \text{s.t.} \quad & \sum_{p \in P} a_{f,p} \cdot x_p \le 1 \qquad \forall\, f \in F \\ & x_p \in \{0,1\} \qquad \forall\, p \in P. \end{aligned}$$

Here, all notation is the same as in (I_{SP}) for modeling the original crew pairing problem in Chapter 2, except for the two new pieces of notation I_p and c_b. Boolean parameter I_p indicates whether pairing p contains a break or not, while constant c_b introduces the associated penalty cost, if so.

As before, we still apply a heuristic to solve this updated set packing formulation. That is, we first use a delayed column generation approach to solve its LP-relaxation, presented in $(L_{SP}*)$ below, and then directly apply mixed integer programming techniques to the

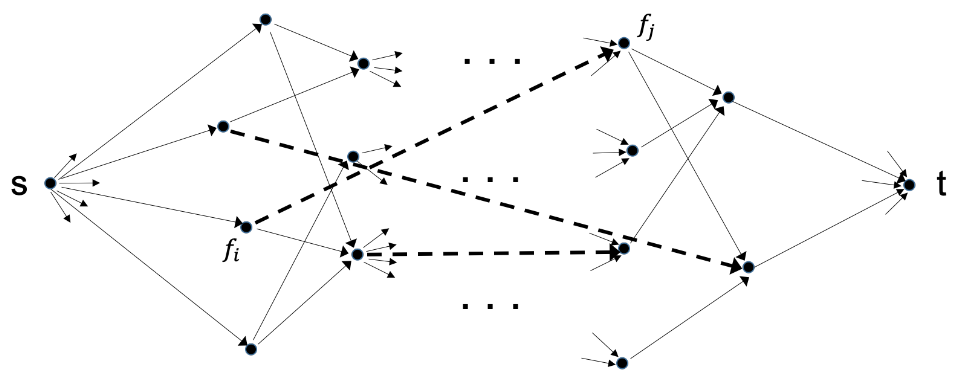

Figure 3.1: The modified flight-based network with break arcs

restricted set packing formulation variation (where the set P is limited to those generated pairings) to produce a final crew pairing schedule.

$$(L_{SP}*) \qquad \begin{aligned} \min_{x_p} \quad & \sum_{p \in P} (I_p \cdot c_b - \sum_{f \in p} c_f) \cdot x_p \\ \text{s.t.} \quad & \sum_{p \in P} a_{f,p} \cdot x_p \ \leq \ 1 \qquad \forall\, f \in F \\ & x_p \ \geq \ 0 \qquad \forall\, p \in P. \end{aligned}$$

To model the pricing problem of the delayed column generation approach to $(L_{SP}*)$, we first introduce an additional set of *break arcs*, denoted as B, to the flight-based network G proposed in 2.5.1. More specifically, for two specific flights, if the idle time between them is no shorter than the minimum duration required for the crew to have a break (i.e., bullet (9) in Section 3.2), then such a break arc is introduced to connect the two nodes they correspond to. Through this modified flight-based network $\bar{G}(V,\bar{A})$, where $\bar{A} = A \bigcup B$, we incorporate the proposed break feature into the crew pairing construction, as now each valid crew pairing, regardless of whether it contains a break or not, corresponds to a specific $s-t$ path in $\bar{G}$. Figure 3.1 provides an illustration of this modified network $\bar{G}$, where dashed arcs represent a subset of those break arcs which are additionally introduced to the network.

Next, to help verify whether a specific $s-t$ path in $\bar{G}$ indeed corresponds to a valid pairing or not, we update the SPPRC model constructed in Section 2.5.2. We keep all of the original seven resources $r_1 - r_7$, as well as each node's upper bound and the REFs for the original set of arcs A on these resources, with exactly the same definition as before. The only thing we need to additionally define is the REFs on the newly introduced break arcs for these seven resources, i.e., how the resource vector on these resources should be updated if the corresponding path is extended through a specific break arc $(i,j) \in B$, which are provided below:

$$\begin{array}{lll} h^1_{i,j}(T) = trv_j & h^2_{i,j}(T) = trv_j & h^3_{i,j}(T) = 1 \\ h^4_{i,j}(T) = T_4 + trv_j & h^5_{i,j}(T) = T_5 - trv_j & h^6_{i,j}(T) = T_6 - trv_j \\ & h^7_{i,j}(T) = T_7 + c_b - c_j - \pi_j. & \end{array}$$

As introduced previously (for the original SPPRC in the previous chapter), $h^r_{i,j}(T)$ here corresponds to the value of the updated resource vector on resource r, after the extension via arc (i,j). trv_j represents the flight time of j, while π_j is the dual value of the constraint corresponding to flight j in ($L_{SP}*$), under the optimal basis for the current iteration of the delayed column generation process.

Note that the values on resources r_1 and r_2 are both reset to trv_j, while the value on r_3 is reset to 1 because a new duty period will start after the break. The consumption of resource r_4 is only augmented by trv_j because the duration of the break is not counted into the span of the whole pairing, nor into the total flying time. By the same logic, the resource extensions on r_5 and r_6 are defined as shown above. Lastly, an additional amount c_b is added to r_7 compared to the resource update through those original arcs in A, as having a break in the corresponding crew pairing introduces an additional penalty cost.

Moreover, to enforce the additional requirements imposed on the potential break, we introduce three extra resources in the SPPRC model, in addition to the original ones $r_1 - r_7$. In other words, these three resources are defined to make sure that, when expanding paths in the udpated network $\bar{G}$, the additional requirements introduced by this break feature (which cannot be guaranteed by the structure of the network) are respected for the validity of the corresponding pairing. The following list provides the definition of each of these new resources, and also specifies how REFs associated with different types of arcs in the network $\bar{G}$ update the consumption of these resources and what upper bounds on these resources are imposed on each node in $\bar{G}$:

r_8: The total number of breaks the current crew pairing has contained so far.

This resource is defined to enforce that the crew pairing can contain no more than one break (i.e., bullet (8) in Section 3.2). Clearly, the value of this resource is initialized to 0 by any arc from the source node s. For every other original arc in A, the REF will not update any consumption on this resource, while the value in the resource vector on this resource will increase by 1 if the corresponding arc is a break arc in B. The upper bound for the consumption of this resource is 1 for each node (except the source node s) in the modified network $\bar{G}$.

r_9: This resource is defined to enforce the requirement specified by bullet (10) in Section 3.2, i.e., if there is a break, it cannot take place until the minimum number of flights (denoted as g_{lb}) in the pairing has been completed.

Regarding the consumption update (i.e., the definition of the REFs on this resource), the value of this resource is initialized to -1 (by any arc pointing from s), and then is decreased by 1 if the corresponding path is extended to a specific flight node through one of the other original arcs in A; otherwise (i.e., the path is extended via a specific break arc in B), the value of this resource is increased by g_{lb}. Lastly, there is no change to the value on this resource when completing the path (i.e., for any specific arc pointing to the sink node t). Value 0 is set as the upper bound for each node (except s) to limit the consumption, which therefore ensures that no crew can have the break before finishing the minimum number of flights.

r_{10}: This resource is defined to enforce the requirement specified by bullet (11) in Section 3.2, i.e., if there's a break, the latest time for it to take place is when the crew completes the maximum number of flights (denoted as g_{ub}) in the pairing.

Similarly, regarding the definition of REFs on this resource, its value is initialized to 1 by any arc from s, while it is kept unchanged for any arc pointing to t (i.e., when completing the path). The value in the resource vector of this resource is increased by 1 if the corresponding path is extended through one of the other original arcs in A; otherwise (i.e., the path is extended via a specific break arc in B), it is multiplied by $(W - g_{ub})$, where W is a theoretical upper bound on the total number of flights that can be contained in a crew pairing. For each node (except s) in $\bar{G}$, we set $(g_{ub}+1)\cdot(W-g_{ub})-1$ as the upper bound for the consumption of this resource. This way, it is ensured that if the corresponding pairing contains a break, there are at most g_{ub} flights in its first flying segment, before the break.

Based on this updated SPPRC model, we can easily verify that the feasibility of a crew pairing is equivalent to the feasibility of its corresponding $s-t$ path in the network $\bar{G}$ with respect to all of the ten resource constraints. Thus, the pricing problem of the delayed column generation approach for this extended crew pairing problem can again be equivalently transformed to solving the following formulation ($PP*$):

$$(PP*) \qquad \min_{p \in \mathscr{P}'} T_7^p.$$

Comparing to the previous formulation (PP) in last chapter, we just replace the set $\mathscr{P}$ with a $\mathscr{P}'$, which now is the set of all feasible $s-t$ paths with respect to all resource constraints in the updated network $\bar{G}$.

According to the updated SPPRC model detailed above, there are no interdependencies among the ten resources with respect to the calculation of their consumption accumulation, while their respective updates via any arc in the network are always either augmentation, subtraction, or multiplication. In other words, all REFs in this updated SPPRC are still coordinate-wise independent, affine functions with non-negative linear coefficients. Therefore, Proposition 2.1 for the original model still holds here, so we can simply apply the previous labeling algorithm, i.e., Algorithm 1, to solve this updated formulation ($PP*$), which thus solves the pricing problem of the delayed column generation framework for this extended crew pairing problem.

Lastly, we introduce three additional improvements to accelerate the convergence of the delayed column generation process for solving ($L_{SP}*$), working together with Improvements (I), (II) and (IV) that we adopted in the finalized approach to the original crew pairing construction in Section 2.6:

(V) We do a warm start for solving the LP-relaxation of the set packing formulation ($L_{SP}*$) by first solving its variation where the break feature is not incorporated, i.e., the crew pairing problem introduced in Chapter 2, to produce a set of non-break pairings.

Clearly, this improvement is guaranteed to provide a valid warm start to the whole column generation process. Furthermore, given the size of the underlying flight-based network and the dimension of the defined resources, solving the SPPRC described in Section 2.5.2 for the original crew pairing problem is much easier than solving this modified one. Particularly, the experimental results in Table 2.5 (the fourth and fifth column, more specifically) have demonstrated that this warm start process can be done in just a handful of minutes, and produce a decent amount of non-break crew pairings for typical real-world intances. In addition, the generated warm start will "weaken" the dual values, because if initialized just with slack variables (without warm start), almost any valid crew pairing will achieve a negative reduced cost during the first few iterations. Therefore, the warm start will result in more paths pruned during the labeling process at an early stage of the column generation. Thus, this warm start generation is expected to improve the overall computational performance.

(VI) During the labeling process, we additionally prune out "useless" paths which cannot possibly be extended to a full pairing that achieves a negative reduced cost.

This is very similar to Improvement (I) we proposed for solving the previous SPPRC for the original crew pairing problem in Section 2.5.4. However, rather than doing pruning by dynamic programming up front, here we need to do backward dynamic programming each time we solve the pricing problem, as the dual values vary for different iterations. More specifically, we calculate for each node in the network the minimum reduced costs starting from this specific node to the sink node considering two different cases, where no break arc is allowed to be traversed in the first case, while at most one break arc can be traversed in the other one. Note that this improvement is also applicable to solving the original SPPRC by the labeling process in the previous chapter. However, we choose not to apply it there because this enhancement requires us to do dynamic programming during each iteration, while the pricing problem for the original crew pairing problem can already be solved in a short time, as demonstrated by experiments on pratical instances.

(VII) We customize the dominance rule to help prune out more paths that are indeed inferior (although their resource vectors may be non-dominated under the conventional definition), when compared with some others during the labeling process.

This improvement is proposed to address the complementary relationship between resources r_9 and r_{10}. That is, since we restrict the location of the break in the pairing to be neither too early nor too late, if the value of the resource r_9 for a specific path is smaller than the value for another path, then it's very likely that the value of r_{10} for the former path is greater than the value for the latter path. By the default definition of dominance, neither of these two paths can dominate the other one. However, in many cases with this circumstance, one path is indeed inferior compared with the other one, and can be pruned out. For example, suppose $T^p = (T_1^p, \ldots, T_7^p,\ 1,\ -1,\ 41)^T$ and $T^q = (T_1^q, \ldots, T_7^q,\ 1,\ -2,\ 42)^T$ are the resource vectors for paths p and q, respectively, where p and q have the same resident node (i.e., end at the same node), and with $T_1^p \leq T_1^q,\ T_2^p \leq T_2^q, \ldots,\ T_7^p \leq T_7^q$. In other words, path p has no greater consumption on the first 7 resources (i.e., $r_1 - r_7$) compared with path q, and both of these two paths have already traversed a break arc, as $T_8^p = T_8^q = 1$; in addition, the corresponding pairing of q has one more flight in total than the pairing of p, while both of these two pairings have the same number of flights in their respective first part, before the break, as $T_9^p = -1,\ T_9^q = -2$ and $T_{10}^p = 41,\ T_{10}^q = 42$. Although by the default definition T^q is not dominated by T^p (because $T_9^p > T_9^q$), path q is actually inferior, since for any feasible extension (q, e), the alternative extension (p, e) will also be feasible while achieving a no worse objective value (i.e., r_7). This is because both p and q already contain a break (thus, any valid extension e cannot include a break arc), and thus we can ignore the three additional resources introduced by the break feature (i.e., r_8, r_9 and r_{10}) in terms of the dominance checking. Since path p has no greater consumption on the other 7 resources (i.e., $r_1 - r_7$) than q, path q is thus indeed inferior.

However, even with these additional improvements, we still find the delayed column generation to converge unacceptably slow. For example, we applied this proposed, modified approach to model and solve the extended crew pairing problem (with the break feature incorporated) for the 606-flight instance described in Section 2.6. It took more than ten hours to solve the pricing problem in the first iteration alone (where more than 2.6 million negative reduced cost pairings were identified). This is mainly a function of the density of the modified underlying network $\bar{G}$ with the associated large number of new arcs added. For this 606-flight instance, the total number of arcs increases from 12,539 to 123,612, which is more than one third the number of arcs in the corresponding complete graph. Since pruning paths also becomes harder due to the increased resource dimension, the labeling process proceeds unacceptably slowly.

To address this computational challenge, we propose an alternative, exact approach based on an arc selection process, which is detailed in the next section.

3.5 An Arc Selection Approach

To model the break in the crew pairing construction, a large number of break arcs are introduced to the underlying network, as the example above suggests, which causes the severe bottleneck for the labeling process. However, not all of those break arcs will necessarily appear in a negative reduced cost pairing during a specific iteration, particularly when the dual values are sufficiently "weakened" after several iterations have been completed. Introducing all such arcs to the network increases run time without any associated benefit. Therefore, we propose to only include (a subset of) those break arcs which are guaranteed to appear in at least one valid, negative reduced cost crew pairing in the network for each specific iteration.

As already mentioned in Section 3.2, for a specific crew pairing with a break to be valid, its first flying task segment, before the break, should satisfy all requirements except the three global requirements imposed on the whole pairing. The same thing applies to the second segment. Then, after concatenating these two parts via the break (with sufficient duration), those global requirements should be collectively satisfied. In addition, to constitute a desirable pairing (for the delayed column generation procedure), these two segments should, together, achieve a negative reduced cost after incorporating the penalty for the break. By this analysis, we develop a 3-step arc assessment process, based on a bidirectional labeling procedure on a tiny sub-network, to determine, for a specific iteration, which break arcs will appear in at least one valid negative reduced cost pairing so that they are beneficial to be included in the network:

- **Step 1.** We model the updated SPPRC (i.e., the one with ten resources in Section 3.4) but on the original (i.e., non-break-arc) network G. Then, we identify and store the set of efficient paths E_f, whose resource vectors are non-dominated, for each flight node $f \in F$ through the proposed labeling process (with the proposed, appropriate speed-up improvements).

- **Step 2.** We reverse the original, non-break-arc network G by flipping the direction of each arc $a \in A$. Then, we define a SPPRC similar to the original one with seven resources on this reversed network, where the resource vector represents the status of the reversed crew pairing that corresponds to the path “backward” from the “sink node” t (see an example in Appendix 3.8.1 for demonstration). Similarly, for each node $f \in F$, we identify and store the set of efficient paths $\bar{E}_f$, from t “backward” to f, through the labeling process with appropriate speed-up improvements.

- **Step 3.** For each break arc $b := (i,j) \in B$, we check if there exists an efficient path $e_i \in E_i$, from s towards i, and an efficient path $\bar{e}_j \in \bar{E}_j$, from t “backward” to j, such that the $s-t$ path, the concatenation $(e_i, b, \hat{e}_j)$, corresponds to a valid crew pairing with negative reduced cost. Here, $\hat{e}_j$ denotes the path $\bar{e}_j$ in the reverted direction in the original network, from j towards t. More specifically, we focus on the values of resources on crew pairing span and cumulative flight time requirements plus the reduced cost in the corresponding resource vectors, denoted as T^{e_i} and $\bar{T}^{\bar{e}_j}$ for path

e_i and $\bar{e}_j$ in the respective SPPRC model described in **Step 1.** and **Step 2.** above, and check whether the summation of the respective values in these two resource vectors meets the target value set for satisfying the corresponding minimum/maximum requirements imposed on the entire pairing. The detail of this process is provided in Algorithm 2 below. If this check is passed (i.e., once such a concatenation is found), then b is proven beneficial to be kept in the network; otherwise, after all concatenations are exhausted, we can discard b for this specific iteration.

Algorithm 2 Arc Check

```
1: function CHECK($b := (i,j)$)
2:     for all efficient paths $e_i$ in $E_i$ do
3:         Denote the resource vector of path $e_i$ in Step 1.'s SPPRC model as $T^{e_i}$;
4:         if $T_9^{e_i} \leq -g_{lb}$ and $T_{10}^{e_i} \leq g_{ub}$ then
5:             for all efficient paths $\bar{e}_j$ in $\bar{E}_j$ do
6:                 Denote the resource vector of path $\bar{e}_j$ in Step 2.'s reversed SPPRC model as $\bar{T}^{\bar{e}_j}$;
7:                 if $p_{min} \leq T_4^{e_i} + \bar{T}_4^{\bar{e}_j} \leq p_{max}$ and $T_6^{e_i} + \bar{T}_6^{\bar{e}_j} \leq l_{min}$ and $T_7^{e_i} + \bar{T}_7^{\bar{e}_j} < -c_b$ then
8:                     return TRUE;
9:     return FALSE
```

Note that the condition in line 4 of Algorithm 2 checks whether the break period takes place at an acceptable position (i.e., bullet (10) and (11) in Section 3.2) in the crew pairing that the concatenation $(e_i, b, \hat{e}_j)$ corresponds to. The three conditions in line 7 respectively check whether the corresponding crew pairing respects the minimum and maximum length (i.e., bullet (5) and (6) in Section 2.2), satisfies the minimum cumulative flight time (i.e., bullet (7) in Section 2.2), and achieves a negative reduced cost.

Figure 3.2 provides a high-level illustration of this 3-step arc assessment process.

Based on this arc assessment process, we develop the following alternative approach (Algorithm 3) to solve the pricing problems ($PP*$) during the delayed column generation approach proposed in Section 3.4, to eventually address the extended crew pairing problem

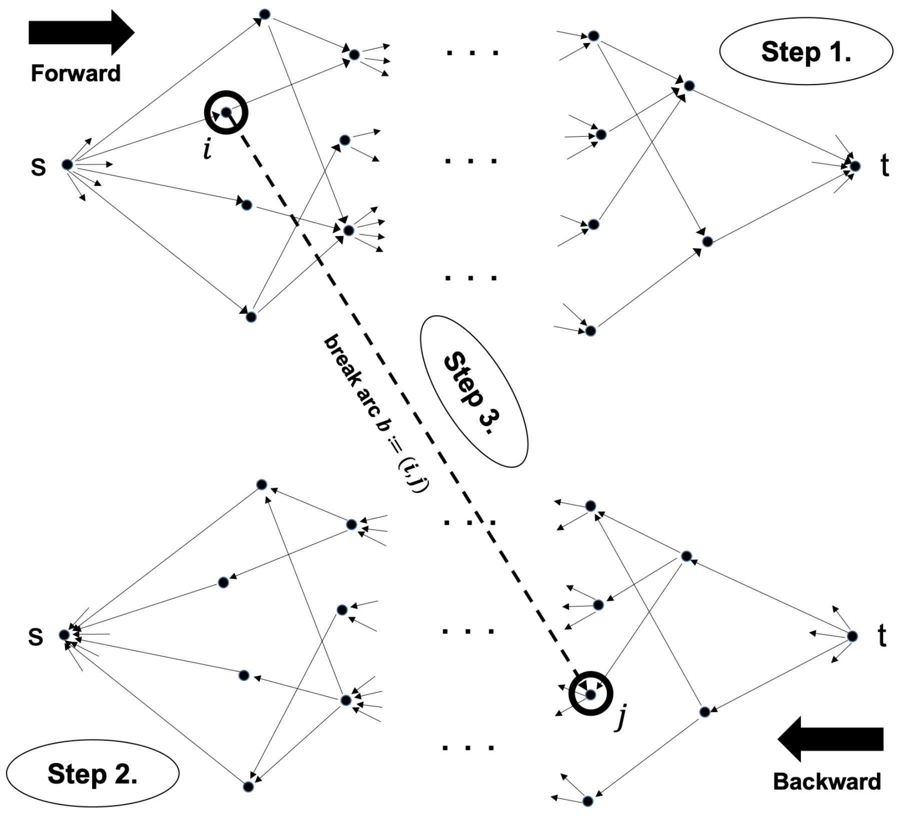

Figure 3.2: A high-level illustration of the proposed 3-step arc assessment process

we consider in this chapter.

Algorithm 3 Arc Selection

```
1: (For solving each specific pricing problem during the delayed column generation process)
2: Follow the 3-step arc assessment process proposed above to check each of the break arcs;
3: Randomly select up to $H$ break arcs that pass the previous check. Keep them in the underlying
   network $\bar{G}$, while temporarily remove the other ones for this specific iteration;
4: Model the updated SPPRC, proposed in Section 3.4, just on this trimmed network;
5: Solve it using the conventional labeling algorithm (Algorithm 1) with all adopted speed-up strate-
   gies (i.e., Improvement (I), (II) and (IV) in Section 2.5.4 plus (V – VII) in Section 3.4);
6: **if** no pairings are identified **then**
7:     Terminate the whole delayed column generation process (i.e., stop the solving of ($L_{SP^*}$));
8: **else**
9:     Add pairings output from the labeling algorithm to the master problem;
10:    The delayed column generation proceeds;
```

We only select up to H of those break arcs (typically controlled within a few hundred) that pass the proposed arc assessment process to keep in the network (line 3 in Algorithm 3) for each specific iteration, instead of all of them. This is to control the size of the finalized, trimmed network and thus ensure the tractability of the corresponding updated SPPRC on it. This additional selection is necessary because a great number of break arcs may pass the check (i.e., appear in at least one valid negative reduced cost pairing), especially during the early stages of the delayed column generation process, while it will not impact the exactness of our approach (see Proposition 3.1 below).

In summary, this new approach can be viewed as that we first temporarily remove all break arcs from the network (i.e., we just work on the original non-break-arc network G), and based on that we perform a bidirectional scanning to help determining (a small subset of) those break arcs that are beneficial to the delayed column generation process and thus restore them back to the network. This arc assessment process is expected to be done in a short time because of the efficiency of the bidirectional scanning, as demonstrated by experiments in Section 2.6 in the previous chapter. Moreover, by this method, the finalized network will be much smaller (sparser) than the original $\bar{G}$, while with respect to solving the pricing problem ($L_{SP}*$), the updated SPPRC model on the trimmed network can be shown to be equivalent to the model on the intact $\bar{G}$ (see more details below). As the numbers specified by the 606-flight instance at the end of Section 3.4 suggest, the final, trimmed network we work on for solving the pricing problem will consist of at most $12,539+H$ arcs, which is much smaller than the $123,612$ arcs in the corresponding $\bar{G}$, and consequently by our experiments, only several seconds are typically required for performing the labeling process on these trimmed networks.

We want to point out that the approach we proposed above is an exact approach, which guarantees the LP-relaxation of the set packing formulation ($L_{SP}*$) to be solved to optimality. This exactness can be shown through the following proposition, whose rigorous proof is provided in Appendix 3.8.2.

Proposition 3.1. A break arc b passes the check in **Step 3.** during our proposed arc assessment process (i.e., Algorithm 2 returns “true”) for a specific iteration, **if and only if** there exists a valid, negative-reduced-cost crew pairing which contains a break that corresponds to b.

More specifically, when the halting criterion (i.e., line 6 of Algorithm 3) is satisfied, it is clear that no valid crew pairing without a break existing in the instance achieves a negative reduced cost. Additionally through this proposition, we know that no break arc could have passed the check during the arc assessment process in this case, because otherwise at least one of such qualified break arcs should be kept in the network, which would lead to a valid crew pairing with a negative reduced cost identified during the labeling process. This implies that there does not exist a valid, negative-reduced cost crew pairing in the instance which contains a break. Therefore, we can conclude that all valid crew pairings have a non-negative reduced cost at this point, and thus the LP-relaxation of the set packing formulation ($L_{SP}*$) has been solved to optimality.

Lastly, there exists another alternative way to solve the pricing problem, based the method proposed above. That is, in addition to applying the original approach to the original crew pairing variation (no break incorporated) to identify a set of non-break pairings, we produce additional crew pairings which each contains a break through trying all possible concatenations during **Step 3.** (i.e., fully enumerating the combinations between the set of “forward” efficient paths and the set of “backward” efficient paths for all break arcs). During this process, besides the identified non-break pairings, we directly add (overall up to K) the corresponding pairings of those who pass the check (i.e., line 4 & 7 in Algorithm 2) to the master problem to proceed, instead of terminating each specific enumeration once a satisfactory concatenation is found, while updating the network accordingly and performing another labeling process on the finalized network once all enumerations are done. Since crew pairings constructed this way are guaranteed to be valid and achieve a negative reduced cost (see the "**only if**" part in Appendix 3.8.2 for the proof), we can ensure that only desirable columns are introduced to the master problem for each specific iteration, and furthermore, by Proposition 3.1 and the analysis above, the exactness preserves. However, we choose to not implement this approach because there could be too many possible concatenations, as thousands of efficient paths may exist in both sets E_i and $\bar{E}_j$, and thus a significant amount of time would be spent to check all combinations. In addition, an unnecessarily large number of desirable pairings would eventually be identified this way, especially during the early stages of the column generation. Therefore, rather than explicitly enumerating pairings that include a break through this brute-force approach, we instead do an arc selection by the proposed check, which can be done in a much shorter time, and then leverage the efficiency of the labeling algorithm on the SPPRC to generate an appropriate number of satisfactory crew parings as described in Algorithm 3.

3.6 Computational Experiments

We implemented the updated model and proposed solution approach using C++ (Visual Studio 2017) with CPLEX (version 12.80) on a 64-bit operating system computer with two 2.10 GHz processors and 128GB RAM. To evaluate its effectiveness for the extended cargo crew pairing problem, we applied it to solve the previous three typical real-world problem instances in Section 2.6 from our cargo airline partner. We present the results in this section.

Table 3.1 below summarizes the basic information of each of these three datase, which is just a copy of Table 2.3, and provided here again simply for an easier reference. Table 3.2 provides some general information of the updated flight-based network $\bar{G}$ we proposed in Section 3.4 together with the full enumeration results for each of the three instances as in

Table 2.4 for the previous problem. Here, the entire set of valid crew pairings are generated by a similar depth-first-search (DFS) approach to the one used in the previous chapter for the original crew pairing problem.

Table 3.1: General information for the 3 instances. "Long-Haul" shows the percentage of flights which have a flight time greater than 4 hours. "Int-Int" refers to flights whose origin and destination are both outside the U.S., while "Dom-Int" refers to flights flying from some place domestic in the U.S. to somewhere outside the country.

Instance	#Flights	Long-Haul	Int-Int	Int-Dom	Dom-Int	Dom-Dom
No.1	606	92.41%	49.84%	23.93%	25.41%	0.82%
No.2	541	92.24%	50.28%	25.14%	24.03%	0.55%
No.3	644	86.96%	47.36%	23.91%	23.91%	4.81%

Table 3.2: A summary of the underlying flight-based network and the full enumeration results for the 3 instances.

Instance	#Nodes	#Arcs	$\lvert A\rvert$	$\lvert B\rvert$	#Valid Pairings	Enum. Time
No.1	608	123,612	12,539	111,073	142,777,637	3day 02hr
No.2	543	97,716	10,113	87,603	79,648,029	1day 21hr
No.3	646	134,907	12,201	122,706	133,208,846	3day 02hr

The numbers show that, although allowing the crew to have a break within the pairing could potentially increase flight coverage significantly, the number of feasible crew pairings, besides the tremendous enumeration time, is too huge for us to even explicitly define the set packing formulation ($I_{SP}*$) in CPLEX. Moreover, the huge number of break arcs (i.e., $|B|$) for modeling this break feature in the crew pairing construction makes the proposed flight-based network $\bar{G}$ extremely dense. All of these require us to again utilize a delayed column generation framework, while in addition, developing new approaches to overcome the density issue, as we've presented in detail in the previous sections.

For all three instances in the following experiments, we treat the excessive cost for operating an uncovered flight by a reserve crew equally across all scheduled flights (like what we did in Section 2.6) and ignore the penalty cost of having a break. In other words, we have $c_b = 0$, while $c_f = 1$ for all flights $f \in F$ in formulation ($I_{SP}*$) and ($L_{SP}*$), and therefore we are simply maximizing the number of flights covered by crew pairings. The reason for this parameterization is exactly the same as it for the experiments on the original crew pairing problem in the previous chapter. That is, our partnered cargo airline views flight coverage as the highest priority at the moment and works with the assumption that having less flights that need to be assigned to reserve crews will result in lower overall costs in practice.

3.6.1 A Benchmark

We consider a simple partial pricing heuristic here to provide a benchmark in order to demonstrate the effectiveness of our proposed arc selection approach in terms of solving our pricing problem for this extended crew pairing problem.

More specifically, for each pricing problem of a specific iteration, we randomly, evenly partition the set of flights scheduled during the planning horizon, and construct the modified flight-based network (with break arcs) as described in Section 3.4 based on each of these two subsets of flights separately. Then, we simply define the updated SPPRC independently on these two "halved" networks, and solve them using the conventional labeling algorithm (Algorithm 1) with the proposed speed-up improvements. We introduce the negative reduced cost crew pairings identified from both networks to the master problem to proceed. If neither of these two networks can provide desirable crew pairings, then we stop and terminate the delayed column generation process.

Please note that the LP-relaxation of the set packing formulation ($L_{SP}*$) is not guaranteed to be solved to optimality when the previous criterion is met, because we skip the possibilities of flight combinations across the two networks. Therefore, this benchmark approach is simply a partial pricing heuristic. However, it can effectively address the described computational challenge, as the runtime of the labeling process is highly dependent on the number of nodes in the network (Warburton, 1987), and from experiments, it will only take around one minute for the conventional approach to label the "halved" network for our typical, practical instances.

We applied this heuristic approach to solve the three instances described previously to obtain the benchmark results, where we set the parameter K in Improvement (II) proposed in Section 2.5.4 equal to $20,000$. In addition, we limit a total maximum 4 hours spent during the whole delayed column generation process for solving the LP-relaxation ($L_{SP}*$) to avoid the tailing effect of its convergence, while we also set a maximum 2 hours on solving the restricted IP (i.e., the set packing formulation $I_{SP}*$ with decision variables limited to those introduced during the column generation process for solving its LP-relaxation $L_{SP}*$). Given the randomness associated with this approach (as we randomly partition the flights each time), we repeated solving each instance independently ten times to avoid "luckily" just reporting the best or the worst case as the benchmark, presuming the heuristic may not perform stably. The corresponding results, summarized by the mean, the standard deviation (SD), and the minimum and maximum of the ten trials for the three instances are respectively provided in Table 3.3, 3.4 and 3.5. Runtimes are all presented in seconds.

Table 3.3: Summarized results for applying the flight partitioning heuristic approach to Instance No.1.

Instance **No.1**	LP-obj	#Iter.	Time	#Pairgs Gen.	IP-obj	Time	Gap(%)	Covrg.(%)
Mean	551.53	129.50	14,400	61,600	494.20	7,200	10.89	81.55
SD	0.70	4.54	0	4,809	4.64	0	1.07	0.77
Minimum	550.70	120	14,400	54,231	487	7,200	9.66	80.36
Maximum	552.70	134	14,400	71,445	500	7,200	12.39	82.51

Table 3.4: Summarized results for applying the flight partitioning heuristic approach to Instance No.2.

Instance **No.2**	LP-obj	#Iter.	Time	#Pairgs Gen.	IP-obj	Time	Gap(%)	Covrg.(%)
Mean	475.28	133.20	6,108.57	46,363	426.90	7,200	9.93	78.91
SD	4.13	29.51	1,330.93	4,393	5.96	0	1.21	1.10
Minimum	467.31	79	3,852.64	40,622	416	7,200	7.60	76.89
Maximum	480.66	178	8,446.83	56,665	439	7,200	11.39	81.15

Table 3.5: Summarized results for applying the flight partitioning heuristic approach to Instance No.3.

Instance **No.3**	LP-obj	#Iter.	Time	#Pairgs Gen.	IP-obj	Time	Gap(%)	Covrg.(%)
Mean	569.37	132.40	13,066.05	58,888	510.30	7,200	10.85	79.24
SD	2.71	22.11	2,328.57	3,425	6.83	0	1.12	1.06
Minimum	563.14	83	7,776	55,027	499	7,200	8.73	77.48
Maximum	572.10	158	14,400	67,331	523	7,200	12.82	81.21

Here, the second column displays the final objective value (actually, the negative objective value, as it can then be interpreted as the number of flights covered) we get for the LP-relaxation of the set packing formulation (i.e., $L_{SP}*$). The number of iterations taken, the total time spent, and the number of valid crew pairings generated during the delayed column generation process are provided in the third, fourth, and fifth column, respectively. The sixth column shows the (negative) objective value we can achieve by solving the restricted set packing formulation ($I_{SP}*$) by CPLEX with just crew pairings generated during the delayed column generation process (i.e., the heuristic approach described at the beginning of Section 3.4), while the corresponding time spent on solving this integer program is presented in the seventh column. The eighth column provides the remaining LB/UB gap of the branch-and-cut process by CPLEX, when the 2-hour time limit has been reached (or when the optimality has been proved). The last column displays the corresponding achieved flight coverage.

3.6.2 Results and Comparisons

The final results of applying our proposed arc selection approach to deal with the pricing problem for solving the three practical extended cargo crew pairing instances are presented (in the corresponding first row) in Table 3.6. Here, we set $H = 250$ in Algorithm 3, i.e., up to 250 qualified break arcs are kept in the network for each specific iteration, where this number was selected based on computational experimentation. As previously, the parameter K in Improvement (II) in Section 2.5.4 is set as $20,000$ for testing all three instances.

We still limit a total maximum 4 hours spent during the whole delayed column generation process, and a maximum 2 hours on solving the restricted original IP. The columns of the table are the same as in the three tables for the benchmark in the previous sub-section. Lastly, to provide a direct comparison, the average value for each attribute achieved by the partial pricing heuristic, i.e., the benchmark approach, in Section 3.6.1 (i.e., row "Mean" in Table 3.3, 3.4 and 3.5) is copied in the parentheses in the corresponding second row. We choose to compare the results against only the mean values from the partial pricing heuristic, because this benchmark approach is shown to be not sensitive to its associated randomness, as demonstrated by the small SD values and the small gaps between the respective minimum and maximum values.

Table 3.6: Computational results for applying the arc selection approach to the three instances. Numbers in the parentheses correspond to the average performance achieved by the benchmark approach. Values in columns LP-obj and IP-obj are shown in negative, presenting the amount of flights covered.

Instance	LP-obj	#Iter.	Time	#Pairgs Gen.	IP-obj	Time	Gap(%)	Covrg.(%)
No.1	563.29	41	7,766	79,730	521	7,200	7.81	85.97
	(551.53)	(129.50)	(14,400)	(61,600)	(494.20)	(7,200)	(10.89)	(81.55)
No.2	492.10	35	4,447	58,448	454	7,200	7.89	83.92
	(475.28)	(133.20)	(6,109)	(46,363)	(426.90)	(7,200)	(9.93)	(78.91)
No.3	584.42	37	7,009	85,050	551	7,200	5.72	85.56
	(569.37)	(132.40)	(13,066)	(58,888)	(510.30)	(7,200)	(10.85)	(79.24)

According to these results, we can first see that the 4-hour limit has never been reached when the delayed column generation process is terminated, which implies that the LP-relaxation ($L_{SP}*$) is indeed solved to optimality by our arc selection approach using just a couple of hours in total. This demonstrates that our proposed approach indeed effectively addresses our computational challenge, as originally the cost of the modified approach in Section 3.4, i.e., the straightforward modification of the original approach proposed in Chapter 2, was over ten hours for a single iteration. By comparing to the benchmark, this arc selection approach is shown to outperform the simple heuristic in terms of both efficiency and solution quality. First of all, a better objective value for the LP-relaxation ($L_{SP}*$) is naturally achieved (the corresponding values in the table are displayed in negative of the true ones, to interpretably represent the amount of covered flights), as this arc selection approach is exact, and it solves each instance to optimality. Furthermore, our arc selection approach needs even fewer iterations and less total time for achieving a better objective value (proving the optimality). This is even true if comparing to the best result (i.e., the respective minimum) achieved by the benchmark heuristic among the repetitions for instances No.1 and No.3, while only a slightly greater total runtime is observed for instance No.2. Consequently, a higher objective value to the original integer program ($I_{SP}*$) can always be provided by our proposed approach through the heuristic method for handling the integrality constraints mentioned at the beginning of Section 3.4. (Note that this heuristic method is workable since only a very small number of feasible crew pairings — less than 0.1% — are generated and explicitly incorporated into the formulation.) More specifically, the arc selection approach achieves around five more percent flight coverage in average. It leads to approximately 85% coverage at the end using in total only a handful of hours, which is a very high coverage value in practice, achieved in a sufficiently short time, and specifically is highly satisfactory to our partnered cargo airline.

3.7 Conclusion

In this chapter, we consider solving an extension of the original crew pairing problem in the previous chapter, where the cargo airline in addition accepts a break to take place in the middle of crew pairings. The incorporation of this feature is necessary because otherwise valid crew pairings are frequently not able to cover sufficient flights for the airline to make profits, due to the nature of the underlying low-connectivity flight network. This break feature is thus introduced, working as a special deadheading method, to relax the original crew pairing problem, in order to greatly boost flight coverage at the end.

A straightforward modification of the previous solution framework proposed in Chapter 2 can make it still applicable to this extended variation. That is, we again modeled the crew pairing problem as a set packing problem, and adopted a delayed column generation framework to handle the exponentially-large number of decision variables in the corresponding formulation. An extra set of break arcs were introduced into the flight-based network to incorporate this break feature, based on which the SPPRC model was updated to enforce the additional requirements on the break to formulate the pricing problem.

Although the conventional labeling algorithm proposed to address the original crew pairing problem can still in theory solve this updated SPPRC model exactly, the extremely high density of the flight-based network, caused by the incorporation of the great number of break arcs, makes it proceed unacceptably slowly if we apply it directly to practical instances, even enhanced by all of our proposed speed-up improvements.

To address this computational challenge, we proposed an arc selection approach to dynamically trim the updated, dense network through a bidirectional search based arc assessment process to temporarily prune most of arcs for each specific column generation iteration. This way, we can equivalently solve the original pricing problem by simply performing the conventional labeling algorithm on a much smaller sub-network, as we've shown that this arc selection is an exact approach. Based on our experiments on real-world instances, this proposed approach was demonstrated to successfully address the described computational challenge, and it can always help producing high quality solutions at the end in a very short time. Moreover, compared to a simple partial pricing heuristic approach, our arc selection approach was shown through experiments to be able to achieve a dominating performance, in terms of both solution quality and runtime effeciency.

Although the details of the proposed arc selection approach and the associated logic are specific to the extended crew pairing problem of our partnered airline, the general approach can be adapted to solve other applications. For example, the idea of the arc assessment process is applicable to any problem where an SPPRC model is formulated and solved (e.g., in traditional passenger crew scheduling problems and vehicle routing problems), while the proposed approach may greatly improve runtime particularly when the underlying network is dense. The exactness of this approach will be preserved if the following three conditions are satisfied: 1) it is applied to a set of mutually exclusive arcs; 2) for every restricted resource defined in the SPPRC, its resource consumption is always accumulated independently; 3) for every restricted resource, either its accumulation is augmentation (or with reset) plus there is a same upper bound on its consumption on every node in the network, or its accumulation is subtraction plus there is only an effective upper bound on its consumption at the end of the network (i.e., on the sink node). Since the SPPRC of many general routing and scheduling problems satisfies these described properties, this proposed approach is expected to be able to help solving many practical applications in an exact manner.

In terms of future research, we would like to reduce the optimality gap when solving the integrality-constrained restricted master of the set packing formulation ($I_{SP}*$), given that a gap of over 5% between the LB and UB was consistently observed in all our experiments (Table 3.6) within the time limit. Potential methods include solving the corresponding *theta body* (i.e., semidefinite relaxation) and additionally generating maximal clique inequalities (Conforti et al., 2014), to help derive a tighter lower bound. In addition, we plan to explore other ways to deal with the integrality constraints in the original set packing formulation, including both other heuristics (e.g., dive-and-price in Ruther et al. (2016); Wei and Vaze (2018)) and exact methods (e.g., branch-and-price framework in Desaulniers et al. (1997); Freling et al. (2004)).

3.8 Appendix

3.8.1 An Example for Step 2. of the Arc Assessment Process in Section 3.5

To demonstrate **Step 2.** of the proposed arc assessment process, let's consider a toy instance, composed of, in total, three flights. Figure 3.3 shows the original non-break-arc flight-based network (i.e., the one used in **Step 1.**) for this instance, where the origin with the departure time and the destination with the arrival time for each flight are displayed in the associated parenthesis. Figure 3.4 illustrates the corresponding reversed network proposed in **Step 2.**.

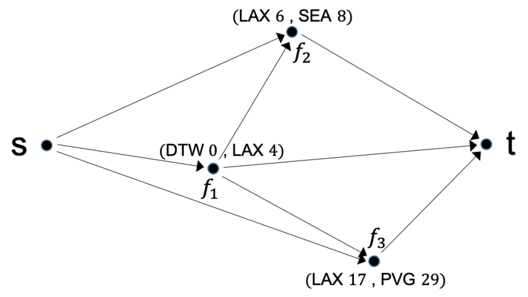

Figure 3.3: The original flight-based network for the 3-flight instance with on break arcs.

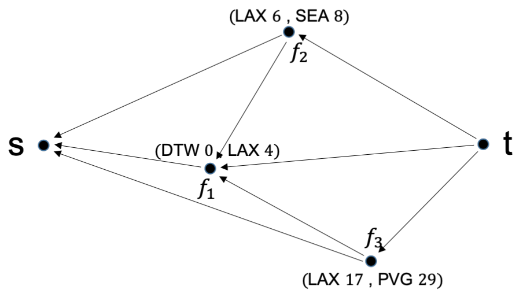

Figure 3.4: The corresponding reversed network in **Step 2.** for the 3-flight instance.

To provide some insights on how the SPPRC is similarly defined on this reversed network, let's calculate the resource vector $\bar{T}^{\bar{p}_1}, \bar{T}^{\bar{p}_2}$ on the first three resources for "backward" paths $\bar{p}_1 := (t, f_2, f_1)$ and $\bar{p}_2 := (t, f_3, f_1)$. Assume the minimum time for a layover is 10 while the minimum time for a day-off is 24. Then for path $\bar{p}_1$, where f_2 and f_1 are in the same duty period since the gap between them is only 2 hours, we have $\bar{T}_1^{\bar{p}_1} = 8$, as the span of the current duty period in the corresponding pairing is 8; $\bar{T}_2^{\bar{p}_1} = 6$, as the cumulative flight time in the current duty period is 6; $\bar{T}_3^{\bar{p}_1} = 1$, as the number of duty periods since last day-off (i.e., the beginning of the pairing) is 1. However, things are different for $\bar{p}_2$, flying f_3 "followed" by f_1, because now the crew will have a layover (but not a day-off) in between. We have $\bar{T}_1^{\bar{p}_2} = 4$, $\bar{T}_2^{\bar{p}_2} = 4$ and $\bar{T}_3^{\bar{p}_2} = 2$ because a new duty period starts after f_3, where f_1 is its first flight and its only flight at this point.

To summarize, the resources defined on this reversed network are the same as those on the original one (no break feature and requirements incorporated) in Section 2.5.2. However, rather than representing the status of the (sub-)pairing specified by a "forward" path from s, upon the completion of its current flight, the resource consumption here is back-tracked from the last flight in the (sub-)pairing specified by a "backward" path from t, upon its current, first one. In other words, the only difference is that the resource consumption here is accumulated through the reversed direction, since for a specific "backward" path in the reversed network, we only know for its corresponding (sub-)pairing which flights are to be completed in the future, rather than the flights the assigned crew has completed in the past. That is, although for a specific arc in the reversed network the point-to flight in reality happens before the point-from flight, we still view the point-to flight as to be completed "following" the point-form flight in terms of the resource consumption calculation. For example, for path $\bar{p}_2$ mentioned above, the values of its resource vector on the first two resources are both 4 (which are defined by the current flight in the corresponding reversed crew pairing, i.e., f_1) instead of 12, the flight time of the actual last flight f_3 in the corresponding pairing.

A more elegant way in terms of implementing **Step 2.** is through creating a mirror of the original instance, where we swap the origin and the destination of each flight while we also swap and negate the corresponding departure time and arrival time. Then, we exactly follow the procedure we introduced in Section 2.5: construct the non-break-arc network, model the SPPRC (with seven resources; no resources defined for the break feature), and solve the corresponding formulation using the labeling algorithm for this mirror instance, which then just accomplish **Step 2.**. Figure 3.5 below provides an illustration of this mirror instance method on the toy 3-flight instance.

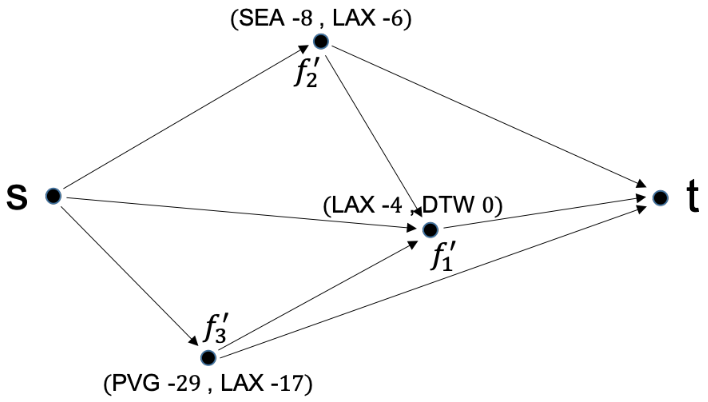

Figure 3.5: The original non-break-arc network for the mirrored 3-flight instance.

3.8.2 A proof for Proposition 3.1

For the **"only if"** part: let $e_i^* \in E_i$ and $\bar{e}_j^* \in \bar{E}_j$ be the two efficient paths that result in the break arc $b := (i, j)$ passing the check in **Step 3.** (i.e., make Algorithm 2 return true). We claim that the $s-t$ path, i.e., the concatenation $p_c := (e_i^*, b, \hat{e}_j^*)$, corresponds to a valid crew pairing (denoted as c) which achieves a negative reduced cost. First, it's obvious that bullet (1*), (8) and (9) in Section 3.2 are respected by pairing c, as they are ensured by the structure of the underlying network. The break arc b corresponds to is guaranteed to take place in the "middle" of c (i.e., bullet (10) and (11)) because the condition in line 4 of Algorithm 2 is passed. According to the way we define the SPPRC in **Step 2.**, the resource consumption is still accumulated through either augmentation, augmentation with reset, or subtraction for all resources, exactly the same as it is achieved in the SPPRC for the original crew pairing problem introduced in the previous chapter but just in a reversed direction. Therefore, with respect to a specific resource, for each path within its two adjacent resets (if applicable), the consumption on this resource accumulated "backward" along this path in the reversed SPPRC is always the same as the amount accumulated "forward" along this path in the original SPPRC. We know that the "backward" path $\bar{e}_j^*$ is feasible in the reversed SPPRC while the break period is long enough so that consumption values for resources r_1, r_2 and r_3 will all be reset. In addition, we know that in both directions the consumption of these three resources is non-decreasing between adjacent resets. Therefore, we can conclude that the corresponding requirements specified by bullet (2), (3) and (4) in Section 2.2 are always respected by pairing c. Based on the same analysis, we also have $T_4^{e_i^*} + \bar{T}_4^{\bar{e}_j^*}$ equals $T_4^{p_c}$, the total resource consumption of path p_c on r_4 under the definition of the updated SSPRC in Section 3.4 (i.e., the total time span of pairing c), while similarly $T_6^{e_i^*} + \bar{T}_6^{\bar{e}_j^*} = l_{min} + T_6^{p_c}$. Lastly, $T_7^{e_i^*} + \bar{T}_7^{\bar{e}_j^*} = T_7^{p_c} - c_b$, since neither the corresponding reduced cost resource value of e_i^* nor $\bar{e}_j^*$ includes the additional cost (resource consumption), i.e., c_b, introduced by the break. Since the three conditions in line 7 of Algorithm 2 are passed, we have that the corresponding crew pairing c satisfies requirements specified by bullet (5), (6) and (7) in Section 2.2, and achieves a negative reduced cost. Therefore, we conclude that c is such a pairing we are looking for.

For the "**if**" part: suppose there is a valid crew pairing with a negative reduced cost which contains a break. Denote the $s-t$ path it corresponds to in the updated network $\bar{G}$ in Section 3.4 as $(e'_i, b, \hat{e}'_j)$, where e'_i is the sub-path from s to i, $b := (i, j)$ is the break arc corresponding to the break period in the pairing, and $\hat{e}'_j$ is the sub-path from j to t. In addition, let $\bar{e}'_j$ be the path in the reversed network in **Step 2.**, from t "backward" to j, that corresponds to $\hat{e}'_j$. Clearly, e'_i is a feasible path in the SPPRC modeled in **Step 1.** of the arc assessment process. So is $\bar{e}'_j$ in the reversed SPPRC in **Step 2.**, because of the features and relationship in terms of the resource consumption accumulation, compared to the original SPPRC, we analyzed in the previous paragraph. Therefore, based on the property presented by Proposition 2.1, we know that there exist a "forward" efficient path $e_i^* \in E_i$ and a "backward" efficient path $\bar{e}_j^* \in \bar{E}_j$ such that $T^{e_i^*} \leq T^{e'_i}$ and $\bar{T}^{\bar{e}_j^*} \leq \bar{T}^{\bar{e}'_j}$. Since the $s-t$ path $(e'_i, b, \hat{e}'_j)$ is feasible with respect to all resource constraints defined in the extended SPPRC (i.e., the one with ten resources, described in Section3.4) with a negative value of r_7, by the same reasoning as in the previous paragraph, the conditions in line 4 and line 7 of Algorithm 2 will be satisfied by resource vectors $T^{e'_i}$ and $\bar{T}^{\bar{e}'_j}$, if directly plugged in. Note that $p_{min} \leq T_4^{e_i} + \bar{T}_4^{\bar{e}_j}$ is equivalent to $T_5^{e_i} + \bar{T}_5^{\bar{e}_j} \leq p_{min}$ for any pair of e_i, $\bar{e}_j$ respectively in the original/reversed SPPRC. Therefore, we can conclude that Algorithm 2 on the corresponding break arc b will return true, at least when e_i^* and $\bar{e}_j^*$ are traversed respectively by the nested loops, which means b will pass the check in **Step 3.**.

CHAPTER 4

A Two-Stage Partial Fixing Approach for Solving the Residency Block Scheduling Problem

4.1 Introduction

In this chapter, we consider constructing a feasible schedule to a large-scale medical residency scheduling problem in which the key decisions are to determine how to assign each resident to different services for different time periods across the academic year. In this section, we provide a brief overview of residency, describe the problem in more detail with respect to our collaborating institution (the University of Michigan Medical School, UMMS), and introduce the basic concepts behind this research. In addition, we present our research motivation as to the need for a new approach to solving this problem.

4.1.1 Residency Programs and the Block Scheduling Problem

In the United States, typical medical training starts with an undergraduate pre-medical program, followed by four years of medical school to complete the M.D. degree. Then, trainees typically spend another three to four years in residency, training and caring for patients under the supervision of more senior physicians before either continuing on to fellowship or beginning fully independent practice.

During residency, these trainees focus on pursuing their educational goals including specialization, while at the same time they provide patient staffing to provide care throughout different units in the hospital and in outpatient settings. Residency scheduling must therefore allocate this scarce resource (i.e., the residents) so as to satisfy multiple objectives related both to training and to patient care.

UMMS provides residency programs in many fields, including Pediatrics (Peds), Internal Medicine (IM), and a hybrid of the two, called Med-Peds (MP). Each of these programs requires its residents at different levels, i.e., Post-graduate Year (PGY) 1, 2, 3 and 4, to complete different *services* during the academic year (from July 1st to June 30th). Here, a service is a specialty in a specific unit in the hospital, for instance Ambulatory (AMB), Emergency Room (ER), etc. The academic year is evenly divided into several time periods, typically 24 or 26, so that each corresponds to a half month or two weeks, and we call each time period a *block*. The *residency block scheduling problem* is thus to assign residents to specific services on specific time blocks, so as to meet training and patient care needs.

The chief residents and program directors at UMMS must construct the annual block schedule for each resident before the academic year starts. As mentioned above, this block schedule should ensure the residents meet their educational requirements while also staffing the hospital to provide sufficient patient care coverage. In addition, before the construction of the schedules, each resident submits a survey to indicate his or her prioritized requests on vacation times and electives. A high quality schedule should consider such preferences at an individual level, and ensure fairness across the residency programs, i.e., no resident receives a significantly easier or more desirable schedule than any other residents for the academic year, while also taking the logistics preferences from the residency programs into account.

4.1.2 Research Motivation

Although the resident block scheduling problem at UMMS has many objective criteria, related both to patient care and resident training and personal satisfaction, in this chapter we focus specifically on the feasibility problem. This problem in turn is the underlying foundation for our approach to supporting UMMS in building an acceptable schedule. In particular, we have implemented an interactive approach whereby we provide feasible solutions to our clinical collaborators (the program directors and chief residents), they provide feedback, and we generate a modified schedule to better match their requests. We repeat this review-and-refine process multiple times until the finalized schedule is satisfactory to all stakeholders.

We use this interactive approach, rather than formulating an objective function with all preferences incorporated and optimizing it, because it is hard to quantitatively formulate some preferences and to properly trade-off the importance of different metrics in the objective function, as they are depending on the subjective judgement of our clinical collaborators. Moreover, many requirements and preferences are subject to change after our collaborators review the current schedule. Therefore, optimizing an objective function is neither sufficient nor effective to handle the preferences from different stakeholders and a satisfactory schedule cannot be expected to be produced by one shot.

Clearly, this interactive approach requires us to be able to generate feasible schedules in a reasonable amount of time, as we will repeat the process multiple times over the course of finalizing a schedule. In our experience, a straightforward, conventional approach to modeling and solving this problem is not viable, with some instances taking as much as days or even weeks to solve. Thus, we are motivated to develop a new approach to identify feasible solutions to this challenging combinatorial problem.

The remainder of the chapter is structured as follows: in Section 4.2, we briefly discuss previous work in the literature on personnel scheduling, with an emphasis on applications to healthcare, and outline the contributions we make to this area. Section 4.3 presents the statement of the problem we are considering here. In Section 4.4, we first provide a high-level summary of the base model — an integer program (IP) for formulating our problem, where a conventional branch-and-cut approach can naturally be used to solve it. We describe the long runtime issue associated with this approach, and briefly analyze its underlying causes. We then propose a two-stage partial fixing approach for more efficiently solving our block scheduling problem, in order to address this computational challenge. Section 4.5 presents computational results on real-world instances. Lastly, in Section 4.6, we conclude and provide thoughts for future work.

4.2 Literature Review

In terms of assignment and allocation of hospital staff and resources, the nurse scheduling problem is studied in the majority of the literature. Miller et al. (1976) developed a mathematical programming model that schedules days-on and days-off for all nurses in a given unit for a given shift across a given several weeks' time horizon. A similar days-on/off scheduling problem was also described in the paper of Weil et al. (1995), but the model in this paper was formulated and solved using constraint programming. Alternatively, a goal programming model was used by Azaiez and Al Sharif (2005) to handle multiple objectives, where functions for measuring deviations from 5 different goals were formulated, weighted and incorporated into the objective. A goal programming formulation was also proposed in the paper of Berrada et al. (1996), and a tabu search method for solving the model was introduced. Moreover, a sequential technique and a equivalent weight technique were also proposed, both of which are able to help generate Pareto-optimal solutions. Brusco (1998) modeled a tour scheduling problem (which is similar to shift scheduling) as a generalized set-covering formulation. The model was presented in the form of a Beale tableau, and solved using Gomory's dual, all-integer cutting plane, which was further accelerated by a customized source row selection rule, objective cuts, and an advanced start point.

Resident scheduling problems in most cases not only need to take the service and staffing coverage requirements into consideration but also the trainees' educational requirements. In the literature, resident shift scheduling problems were found more frequently discussed compared with block or rotation scheduling. Cohn et al. (2009) considered a problem assigning residents to three different hospitals simultaneously. A mixed integer programming (MIP) model was formulated, based on which the authors used a three-phase interactive approach to resolve the multi-objective issue arising from the problem. Güler et al. (2013) proposed a goal programming model to enforce the hard rules while penalizing the violations of soft constraints. Analytical hierarchy process (AHP) was used to quantify the weights of the deviation from different goals in the objective function. A similar model was also presented by Topaloglu and Ozkarahan (2011). They used a hierarchical method to solve their model, which considers only a single metric each time by the order of their priority. Sherali et al. (2002) solved a night shift scheduling problem using a MIP formulation. They proved that by minor relaxations, the formulation will become a bounded variables linear network flow programming problem, which will be totally unimodular.

Regarding resident block/rotation scheduling, Guo et al. (2014) proved that the basic resident scheduling feasibility problem is NP-complete. The paper of Franz and Miller (1993) seems to be the first one in the literature to solve a resident rotation scheduling problem. The authors formulated the problem as a MIP, which was then solved by a rounding heuristic. Bard et al. (2016) focused on assigning clinic sessions in different weekly templates to construct annual monthly block schedules for residents. A MIP formulation was first provided. Then, three heuristics, including a local branching technique, were proposed and used to generate high-quality schedules. Another paper written by these authors expanded the previous problem to schedule construction for three hospitals simultaneously (Bard et al., 2017). A similar MIP formulation was built and a trial-and-errors heuristic was used to produce high-quality schedules. In addition, another MIP model was formulated to help figure out the most-diversified subset of a given number of optimal solutions by minimizing the greatest pairwise similarities.

From the above, problems are most commonly formulated as a (mixed) integer program, which will then be directly solved by some branch-and-bound methods or heuristics to generate solutions. But when it comes to dealing with a large-scale scheduling problem, a column generation formulation, solved through a branch-and-price (B&P) framework, could be a more powerful method, and is also widely used. For most of those papers in the literature, the master problem's decision variables correspond to feasible schedules with respect to individual residents or nurses (Jaumard et al., 1998; Maenhout and Vanhoucke, 2010; Brunner and Edenharter, 2011). However, Belien and Demeulemeester (2006) provided a formulation where each column in the master problem corresponds to a feasible assignment pattern for a specific activity (rotation). Detailed computational results about the comparison between these two different formulation schemes can be found in another paper by them (Beliën and Demeulemeester, 2007). Regarding the pricing problem, it can be formulated as a restricted shortest path problem (Belien and Demeulemeester, 2006; Beliën and Demeulemeester, 2007) or a higher-dimensional resource-constrained shortest path problem (i.e., SPPRC) (Jaumard et al., 1998; Maenhout and Vanhoucke, 2010), and then solved via a dynamic programming and/or labeling approach. It may also be efficient enough to just formulate the pricing problem as a MIP and solve it directly (Brunner and Edenharter, 2011). In terms of the branching during the B&P for solving the master problem to integrality, Maenhout and Vanhoucke (2010) and Belien and Demeulemeester (2006), respectively, proposed and compared different branching strategies as well as different variable selecting strategies. A handful of speed-up techniques, including Lagrangian dual pruning and reduce cost fixing, were also proposed and used in their papers.

Constraint programming (CP) is another method, independent of any previous MIP/IP framework, commonly-used for solving personnel scheduling problems in healthcare (Weil et al., 1995; Cheng et al., 1997; Chan et al., 1998; Trilling et al., 2006; Rahimian et al., 2017). By utilizing effective global constraints and propagation mechanisms, CP has been demonstrated to be efficient on finding feasible solutions to large-scale combinatorial problems, which can also be easily generalized to solve optimization versions. Besides CP, more and more research nowadays focuses on applying genetic algorithms to construct personnel schedules and/or timetables (Aickelin and Dowsland, 2004; Aickelin et al., 2008; Adamuthe and Bichkar, 2011; Leksakul and Phetsawat, 2014; Syberfeldt et al., 2015). They developed and experimented with different selection methods and crossover and/or mutation approaches, to rapidly produce high-quality solutions.

Lastly, many papers in the literature considered producing more reliable and globally higher-quality schedules through integrating multi-stage decision-making phases together. Kim and Mehrotra (2015) addressed a nurse shift scheduling problem, where the staffing level for a 12-week horizon is first determined 6 weeks ahead, while shifts can be further added and deleted at the beginning of each week for the following week over the 12-week period when a better understanding of the demand is available. The problem was formulated as a two-stage stochastic integer program, while the second stage was convexified by mixed-integer rounding inequalities. A thin direction branching strategy, an aggregation of the Bender's cut from different scenarios, and a modified L-shaped method were used to achieve a better computational performance. An employee timetabling problem was solved by Detienne et al. (2009), which is to determine the working pattern for each employee first, and then for each working period in the assigned patterns, determine the corresponding qualification that should be used for satisfying coverage requirements. A matching-based cut generation heuristic approach was proposed to solve the problem, which does not require any third party solver. Guyon et al. (2010) integrated an employee timetabling problem with its associated production scheduling problem, which naturally resulted in a two-stage decision-making structure. That is, effective working periods are first set to employees by working pattern assignment, and then competence matching and jobs processing schedule are determined. Besides a conventional Bender's decomposition, another decomposition scheme with a different cut generation process was proposed, which was developed based on a maximum flow model.

4.2.1 Contributions

Compared with shift scheduling, constructing annual block schedules has received significantly less attention, as discussed above. Our research contributes to filling this gap. Furthermore, we have to consider the schedule construction for three residency programs simultaneously because residents from the hybrid program MP share responsibility for covering units in both of the other two, IM and Peds. As a result, the size of our problem is much larger than problems typically considered in the literature. We need to assign approximately 250 residents to around 100 services and sub-services over the year, rather than a few dozen.

A key contribution of our work is in proposing a novel solution framework to address the computational challenge of solving this large-scale problem, as conventional approaches like branch-and-cut by a MIP solver are shown to be insufficient in practice.

Although there is not a natural two-stage decision-making structure, we intentionally partition our decision process into two stages. We first consider accomplishing the assignment of a small number of "picky" services, whose associated constraints already make their assignments highly restricted, and then try completing the remaining pieces of the puzzle after partially fixing these assignments. We develop several cut generation mechanisms to prune off the current unacceptable fixing once infeasibility arises, which therefore ensures this solution framework is an exact method. We verify the effectiveness of our proposed approach by carrying out computational tests on real-world instances. Lastly, the flexibility of our approach allows it to be applicable to other applications, particularly to combinatorial feasibility problems in personnel scheduling and vehicle routing.

4.3 Problem Statement

In order to constitute a valid schedule for residents, several requirements from different perspectives, e.g., residents' education, patient care coverage for hospital units, the administrative logistics of the residency programs and so on, must be satisfied as discussed previously. More specifically, the requirements and rules we consider for constructing the resident block schedule at UMMS can be categorized into the following four groups:

1. **Basic Assignment Rules**: Each resident must be assigned to exactly one service for each block (i.e., time period). These rules ensure a complete schedule structure.

2. **Resident Education Requirements**: Each resident education requirement is defined by a resident, a set of services, a set of time periods, a lower bound, and an upper bound. It says that the total number of blocks in the given time period set, during which the resident is assigned to services in the given service set, should be greater than or equal to the given lower bound but less than or equal to the given upper bound.

For example, a possible requirement is: Resident-1 should complete service Cardiology (Cards) or VA Wards (VW) cumulatively at least 2 but no more than 4 blocks during the whole academic year.

3. **Service Coverage Requirements:** Each service coverage requirement is defined by a set of residents, a set of services, a set of time periods, a lower bound, and an upper bound. It says that the total cumulative number of blocks that the given set of residents are assigned to the given set of services during the time periods in the given set must be greater than or equal to the given lower bound but less than or equal to the given upper bound. For example, a possible requirement is: service Pediatric Emergency Room (PER) requires at least 6 but no more than 7 residents from program Peds or MP in total during the block of the first half of August. For another example: the total number of blocks that PGY-1 residents from program Peds or MP are assigned to service General Inpatient Wards (General) or Pediatric Cardiology (MP-PedsCards) cumulatively across the whole academic year should be at least 200 but no more than 245.

4. **Miscellaneous Rules:** This group contains all of the additional constraints needed to ensure a feasible schedule. For example, for any resident from the MP program, he/she cannot be assigned to service General Medicine (GM) before completing at least one block of service Hospitalist (HOS) or Pediatric ICU (PICU). For another example, the number of residents assigned to service Pediatric Hematology/Oncology (PHO) must be kept the same within each month during the academic year (i.e., the same for each pair of blocks within the same month). For the complete list of these miscellaneous requirements by type, please refer to the notation section in Appendix

4.7.1, where we explain each type and its associated structure in detail.

The objective of our problem is to construct a resident block schedule so that all of the above four groups of rules are satisfied.

4.4 Solution Approach

In this section, we first describe the base model, a pure integer program (IP), for formulating our resident block scheduling problem. The conventional branch-and-cut approach can be applied to solve this model. However, for our practical instances, this approach is shown to perform unacceptably slowly even using a cutting-edge commercial solver. Given this, we present a high-level analysis of the potential causes of this performance issue. Then, we propose a novel two-stage partial fixing approach to address this computational challenge, and thus significantly speed up the construction of a feasible resident block schedule.

4.4.1 The Base Model

The series of primary decisions we need to make is that, for each resident in UMMS residency programs, for each service, and for each block during the academic year, whether this resident will be assigned to this service for this block or not. By accordingly defining this set of binary decision variables, we can formulate our resident block scheduling problem as an integer linear program. The complete formulation is provided in Appendix 4.7.1, and here, without loss of generality, we simply write it in the following general form for reference:

$$\text{(IP)} \qquad \begin{aligned} \max \quad & 0 \\ \text{s.t.} \quad & Ax \le b \\ & x \text{ integer.} \end{aligned}$$

Note that the objective function in (IP) is maximizing 0, as our goal is to find a feasible schedule to our resident scheduling problem. The linear constraints $Ax \leq b$ enforce all of the four groups of requirements introduced in Section 4.3, and the variables bounds, i.e., $0 \leq x \leq 1$ needed to ensure that all decisions are binary, are incorporated into this linear system as well.

Naturally, we can apply the commonly-used branch-and-cut method to solve this model. However, for our instances, this approach may take a few days or even over a week, which is too slow to make the interactive, review-and-refine construction framework perform effectively. Therefore, we set out to develop a more efficient approach to address our resident block scheduling problem.

4.4.2 The Causes of the Slow Computational Performance

Before we present our proposed approach to address the described computational challenge, we first give a high-level analysis of the root causes of this long runtime issue for constructing our block schedule.

First and foremost, the slow performance is due to the huge size of our instances. More specifically, there are approximately 250 residents in total across the three residency programs at UMMS, while there are around 100 different services, and 24 (or 26) blocks (i.e., the academic year is evenly divided into half months or two-week periods). These numbers together specify a huge number of possible combinations. As a result, the corresponding formulation (IP) typically consists of roughly 1 million decision variables and 2 million constraints, whose size is too large for its corresponding LP-relaxation to be solved in a short time (please find more details in Computational Experiments, Section 4.5). Since a large number of such LP-relaxations must be processed during a traditional branch-and-cut procedure, this long runtime eventually leads to unacceptably slow performance.

Secondly, unlike traditional nurse shift scheduling problems, where only coverage requirements need to be taken care of, we have to, in addition, satisfy individual person's educational requirements. The coordination between these two types of requirements simultaneously introduces a significant amount of complexity to our problem. More intuitively, completing our scheduling tasks here can be visualized as filling the following table (Figure 4.1), where each row corresponds to each resident while each column corresponds to each block, and we need to place the service names into all of its cells. The resident education requirements (Group 2 in Section 4.3) respectively impose restriction on filling service names in each row while the service coverage requirement (Group 3) impose restriction on columns. We need to complete this table coordinating these horizontal restrictions and vertical restrictions, which is much more complicated than dealing with only one single direction (e.g., the vertical one for nurse shift scheduling). By this visualization, solving our residency block scheduling problem can also be analogized to solving an advanced Sudoku game. However, our problem is much more difficult, because of both the number and the complexity of the horizontal and vertical restrictions. Particularly, these requirements of our problem apply not just to individual columns or individual rows, but also to subsets of rows and subsets of columns.

	Block 1	Block 2	⋯	Block T
Resident 1	**ER**	*TBD*	⋯	**PHO**
Resident 2	**AMB**	**AMB**	⋯	*TBD*
⋮	⋮	⋮	⋱	⋮
Resident R	*TBD*	**HOS**	⋯	*TBD*

Figure 4.1: Visualization of residency block scheduling.

Lastly, the slow computational performance is often caused by the significant amount of symmetry in our problem. More specifically, some residents (or blocks) are locally interchangeable in terms of their respective assignments. Consequently, for any given fractional (vertex) solution at a specific node in the branch-and-bound tree, there may exist a very large number of other fractional (vertex) solutions that can be formed by a simple permutation on any subset of the fractional "assignments" on those residents (or blocks). This means it will be extremely unlikely that we are lucky to obtain an integral solution directly by solving a specific node among the top layers. We instead need to dive deep into the branch-and-bound tree before we can reach a "leaf node." On the other hand, if we encounter an infeasible node along a specific path in the branch-and-bound tree, then there may exist a very large number of other similar paths (e.g., paths that are identical under permutation of the assignments on those locally interchangeable residents or blocks) which leads to the same infeasibility. For instance, a path that assigns resident r_1 to service s_1 at block t_1 and assigns resident r_2 to service s_2 at block t_2, is different from another path which assigns r_2 to s_1 at t_1 while r_1 to s_2 at t_2. However, they both could potentially lead to an infeasible node with exactly the same underlying root cause, if residents r_1 and r_2 are locally (i.e., at block t_1 and t_2) interchangeable across all of their involved requirements (which is very likely if they are at the same level and from the same residency program). Therefore, we may repeatedly encounter the same infeasibility along different paths many times, before we can eventually jump out of the loop and find a valid path which specifies a feasible solution.

4.4.3 A Two-Stage Partial Fixing Approach

It is clear that if only Basic Assignment Rules and Resident Education Requirements (i.e., Group 1 & 2 in Section 4.3) are enforced, the corresponding IP formulation will become almost trivially easy to solve, since the problem can be decomposed to construct a feasible schedule for each individual resident separately, in addition to the significantly reduced formulation size. Furthermore, we have observed in our computational experiments that if the requirements in the Miscellaneous Rules (i.e., Group 4) are also incorporated, the corresponding IP model can still be solved in a relatively short time. This is expected as the coordination between the educational requirements and coverage requirements is actually the most complicated puzzle we need to solve for constructing our resident block schedule as discussed previously. Table 4.1 below provides the respective formulation size and runtime results for these (sub-)problems using a real-world instance. Intuitively, based on this analysis, we can use a cut generation procedure, where we restore, step by step, coverage requirement constraints that are violated by the current solution of the corresponding relaxed sub-problem. However, this approach does not work in practice because a large number of iterations will be required, while the model will eventually grow to a comparable size to the original formulation.

Table 4.1: The IP formulation size of the different (sub-)problems of a real-world instance, and the corresponding runtime by CPLEX.

(Sub-)problem	Group 1 & 2	Group 1, 2 & 4	Original (All 4 Groups)
#Rows	32,998	2,650,722	2,654,927
#Columns	617,500	1,332,500	1,332,500
Runtime	**7 seconds**	**33 seconds**	$>$ 1 **week**

Our idea is to complete the feasible schedule construction through a two-stage decision-making process, reducing the problems we need to consider at each step to an acceptable size. During the first stage, we focus on accomplishing the assignments of a small subset of services, by solving a relaxed sub-problem where only a small number of service coverage requirements that are relevant to these selected, small subset of services are incorporated. In our computational experiments, we have found that this relaxed problem can be solved quickly. Once we get a feasible schedule to this sub-problem, we partially fix the assignments of those selected services through introducing additional constraints to the original formulation. Then, we try accomplishing the assignments of the remaining services with the goal of producing a feasible schedule through solving this updated formulation (with all original and these additional constraints) during the second stage. Since a significant portion of cells in the scheduling table (Figure 4.1) will be filled, we have observed that this updated formulation during the second stage can also be solved in a relatively short time. Therefore, this proposed two-stage solution framework is expected to significantly reduce the runtime for solving our block scheduling problem, assuming the assignment fixing of those selected services, as the results from the first stage, is acceptable (i.e., will not result in infeasibility for the second stage).

Mathematically, we can describe our proposed two-stage partial fixing approach via the formulation below. Here, given the selected, small subset of services L, we divide the decision variables x into two parts, $x := (x_p, x_q)$, where x_q correspond to the variables that these selected services L are involved in. Initially, we partition the linear constraints $Ax \leq b$ accordingly into two groups as well, denoted as $A^1x \leq b^1$ and $A^2x \leq b^2$, respectively. The second linear subsystem $A^2x \leq b^2$ consists of constraints which enforce the service coverage requirements that are not related to any of those selected services in L. That is to say, subsystem $A^1x \leq b^1$ is comprised of all of the remaining constraints, including those for the other three groups of requirements (except service coverage) in Section 4.3, the variable bounds (i.e., $0 \leq x \leq 1$), and the ones for service coverage requirements where at least one of the selected services L is involved.

Stage 1:

(SG1)

$$\begin{aligned} \max_{x:=(x_p,x_q)} \quad & f(x) \\ \text{s.t.} \quad & A_p^1 x_p + A_q^1 x_q \leq b^1 \\ & x \text{ integer.} \end{aligned}$$

Stage 2:

(SG2)

$$\begin{aligned} \max_{y:=(y_p,y_q)} \quad & 0 \\ \text{s.t.} \quad & A_p^1 y_p + A_q^1 y_q \leq b^1 && (\pi_1) \\ & A_p^2 y_p + A_q^2 y_q \leq b^2 && (\pi_2) \\ & x_q^* - y_q \leq 0 && (\rho) \\ & y \text{ integer.} \end{aligned}$$

Note that we have introduced an objective function $f(x)$ in the first stage's model (SG1). The purpose of doing so is that we want the solver to provide us with a solution that leaves sufficient room for the assignments of the rest of the services (i.e., those not in L), rather than giving us an arbitrary feasible solution. Incorporating an objective function is computationally feasible because we should be able to very easily get a feasible solution to (SG1) as analyzed above, while we will set an acceptable time limit for solving this optimization model, since it's not necessary to attain its optimality. For the same purpose, we also introduce a small number of auxiliary constraints, designed based on the unenforced constraints $A^2x \leq b^2$, to the first stage's model to reduce the possibility that the second stage will become infeasible. The details on the design of the objective function $f(x)$ and these auxiliary constraints are provided in Appendix 4.7.2.

For the formulation (SG2) in stage 2, $x^* := (x_p^*, x_q^*)$ denotes the solution to the first stage's model. Constraints $x_q^* - y_q \leq 0$ ensure that any assignment on the selected services L, specified by the resulted schedule from the first stage, must be maintained (i.e., if the value in x_q^* is 1, then the corresponding value for the variable in y_q must be 1, given all variables are bounded between 0 and 1 by $A^1y \leq b^1$). On the other hand, if a specific assignment is not made during the first stage, we do not prohibit this assignment, and allow it to be made if all other constraints can be satisfied. This is why we call it a partial fixing approach.

This proposed solution framework is very similar to the well-known two-stage stochastic program, but they differ in the following aspects. First, there is no uncertainty in our problem. Thus, there is only one possible realization of the second stage. Second, there is not a structural or temporal difference between the stage one and stage two decisions, but rather we partition solely for the sake of computational efficiency. Lastly, our goal is not to find a first stage solution which optimizes the objective function $f(x)$. In fact, $f(x)$ is viewed more as an auxiliary function, and we instead focus on ensuring the feasibility of the second stage, whose solution specifies the actual schedule we are looking for.

In the case that the second stage formulation (SG2) is infeasible because of an invalid partial fixing on variables y_q (i.e., the assignments specified by x_q^* on the selected services L is infeasible), we require some mechanism to generate cuts to prune off the current unacceptable first stage decisions (i.e., prune off x^*) and then start over, similar to how we generate Bender's cuts to solve the traditional two-stage stochastic program. The easiest way to prune the current fixing on variables y_q is adding the corresponding no-good cut (Balas and Jeroslow, 1972) to the first stage's model. More specifically, given x_q^* from the final solution to the first stage, we add the following constraint to its formulation (SG1):

$$\text{(No-good)} \qquad \sum_{i:\ (x_q^*)_i=1} (x_q)_i \ \leq\ \sum_i (x_q^*)_i - 1.$$

However, from our experiments, this no-good cut cannot provide a consistent computational performance. That is, for some instances, applying this approach can help to generate a feasible solution to our block scheduling problem in a very short time, but for some other instances, it fails to find a solution within the time limit. This unstable performance is expected, given that a very large number of assignment symmetries may exist as analyzed in Section 4.4.2. In other words, although the current unacceptable assignments on those selected services in L can be avoided by (No-good), similar assignments under permutation of locally interchangeable residents and/or blocks can lead to virtually the same infeasibility during the second stage, which is not prohibited by this cut. Therefore, some more robust cut generation mechanisms are required.

To effectively prune off the current solution and similar ones, we alternatively propose two cut generation mechanisms based on two different cases, depending upon whether the LP-relaxation of the second stage's formulation (SG2) is infeasible too or not. For both cases, we will identify a small number of service coverage constraints that are not enforced in the first stage model (i.e., those among $A^2x \leq b^2$), and that “cause” the current infeasibility during the second stage, and bring them back to the first stage. Then, we accordingly update the partitioning of the constraints in the original linear system, i.e., we remove the identified constraints from subsystem $A^2x \leq b^2$ and add them to the subsystem $A^1x \leq b^1$. Then, we repeat the solution process. Note that the objective function and the auxiliary constraints in (SG1) will not be updated during the progress of this iterative process in our current design. We leave this potential enhancement for future research. Also note that the partitioning of decision variables, $x := (x_p, x_q)$, will remain unchanged during this iterative process, as the services whose assignments will be partially fixed during the second stage are always just those in L. For example, suppose service AMB is the only selected service in L (i.e., $L = \{\text{AMB}\}$). Thus, the coverage constraints in subsystem $A^1x \leq b^1$ at the beginning should only be those where service AMB is involved. Then, as we iterate through the first and second stages based on our proposed cut generation mechanisms, this subsystem $A^1x \leq b^1$ evolves, and additional coverage constraints which do not explicitly impose restrictions on AMB but on some other services, say, GM and HOS, will be incorporated into it and enforced during the first stage. However, services GM and HOS will not be added to set L, and only the assignments on service AMB will be partially fixed during the second stage each time. The detailed design and analysis of our proposed cut generation mechanisms for the described two cases, together forming an alternative to the no-good cut approach, are respectively presented in the following two sub-sections (Section 4.4.3 Case 1 and 4.4.3 Case 2).

Figure 4.2 below outlines the whole solution framework we propose for identifying a feasible solution to our resident block scheduling problem.

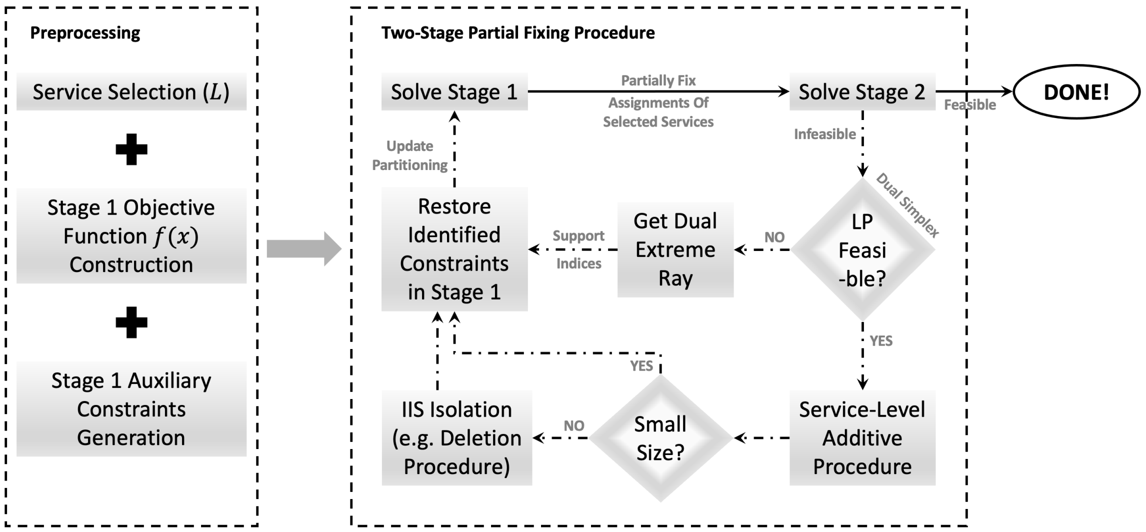
Preprocessing
Service Selection (L)
Stage 1 Objective Function $f(x)$ Construction
Stage 1 Auxiliary Constraints Generation
Two-Stage Partial Fixing Procedure
Solve Stage 1
Partially Fix Assignments Of Selected Services
Solve Stage 2
Feasible
DONE!
Infeasible
Dual Simplex
LP Feasi -ble?
NO
Get Dual Extreme Ray
Support Indices
Restore Identified Constraints in Stage 1
Update Partitioning
YES
Service-Level Additive Procedure
Small Size?
YES
NO
IIS Isolation (e.g. Deletion Procedure)

Figure 4.2: A bird's-eye view of the proposed solution framework.

Cut Generation Case 1 — Infeasible LP-relaxation

We first consider the scenario that not only is the second stage formulation (SG2) infeasible (given the first stage solution x^*), but also so is its LP-relaxation, which is provided in (R-SG2). Let π_1, π_2 and ρ be the dual variables corresponding to the first and second group of constraints and the partial fixing constraints, respectively. Then, we have the dual formulation of the second stage's LP-relaxation in (D-SG2).

(R-SG2)

$$\begin{aligned} \max_{y:=(y_p,y_q)} \quad & 0 \\ \text{s.t.} \quad & (\pi_1) \quad A_p^1 y_p + A_q^1 y_q \le b^1 \\ & (\pi_2) \quad A_p^2 y_p + A_q^2 y_q \le b^2 \\ & (\rho) \quad x_q^* - y_q \le 0. \end{aligned}$$

(D-SG2)

$$\begin{aligned} \min_{(\pi_1,\pi_2,\rho)} \quad & \pi_1 b^1 + \pi_2 b^2 - \rho x_q^* \\ \text{s.t.} \quad & \pi_1 A_p^1 + \pi_2 A_p^2 = 0 \\ & \pi_1 A_q^1 + \pi_2 A_q^2 - \rho = 0 \\ & \pi_1, \pi_2, \rho \ge 0. \end{aligned}$$

Note that **0** is a feasible solution to this dual problem (D-SG2). Thus, it is unbounded when the LP-relaxation of the second stage problem (R-SG2) is infeasible (i.e., this Case 1). Suppose we apply the dual simplex algorithm to solve the LP-relaxation (R-SG2), and suppose $(\pi_1^*, \pi_2^*, \rho^*)$ is the dual extreme ray provided by the solver which proves its infeasibility. In other words, $(\pi_1^*, \pi_2^*, \rho^*)$ is a feasible solution to the dual (D-SG2), and satisfies $\pi_1^* b^1 + \pi_2^* b^2 - \rho^* x_q^* < 0$. Denote the constraints among the second group in (R-SG2) whose corresponding dual values in π_2^* are positive as $A_p^{2+} y_p + A_q^{2+} y_q \leq b^{2+}$. Then, we add constraints $A_p^{2+} x_p + A_q^{2+} x_q \leq b^{2+}$ to the first stage, and update the constraint partitioning of the original linear system $Ax \leq b$ through $\{A^1 x \leq b^1\} \leftarrow \{A^1 x \leq b^1\} + \{A^{2+} x \leq b^{2+}\}$, while $\{A^2 x \leq b^2\} \leftarrow \{A^2 x \leq b^2\} \backslash \{A^{2+} x \leq b^{2+}\}$.

By the theorem provided by Gleeson and Ryan (1990), we know that the support of the dual extreme ray $(\pi_1^*, \pi_2^*, \rho^*)$ specifies an irreducible infeasible subsystem (IIS) for the primal formulation (R-SG2). Therefore, $\pi_2^* \neq 0$, so that the identified set of constraints $\{A^{2+} x \leq b^{2+}\}$ is non-empty, because otherwise (x_p^*, x_q^*) should be an infeasible solution to the previous first stage problem. On the other hand, by the described cut generation method above, it can be seen that the previous solution x_q^* can be prohibited from the current, updated first stage. Furthermore, the following theorem shows that this cut generation approach performs better than the traditional Bender's approach which is widely applied to two-stage stochastic programming.

Theorem 4.1. *When the LP infeasibility arises during the second stage, the cuts $A_p^{2+} x_p + A_q^{2+} x_q \leq b^{2+}$ generated by our approach form a tighter first stage formulation than the Bender's feasibility cut $\pi_1^* b^1 + \pi_2^* b^2 - \rho^* x_q \geq 0$ does.*

Proof. Given our cuts $A_p^{2+} x_p + A_q^{2+} x_q \leq b^{2+}$, we have $\pi_2^{*+} \left(A_p^{2+} x_p + A_q^{2+} x_q\right) \leq \pi_2^{*+} \cdot b^{2+}$, where π_2^{*+} are the corresponding positive entries in π_2^*. Since the corresponding dual extreme ray, which proves the infeasibility, satisfies $(\pi_1^*, \pi_2^*, \rho^*) \geq 0$,

$$(4.1) \qquad \pi_2^* \left(A_p^2 x_p + A_q^2 x_q\right) = \pi_2^{*+} \left(A_p^{2+} x_p + A_q^{2+} x_q\right) \leq \pi_2^{*+} \cdot b^{2+} = \pi_2^* \cdot b^2,$$

as the other coordinates in π_2^*, except those specified by π_2^{*+}, are all valued 0.

In addition, since $(\pi_1^*, \pi_2^*, \rho^*)$ is an extreme ray to (D-SG2), we know that $\pi_1^* A_p^1 + \pi_2^* A_p^2 =$

0, and therefore for any x_p, we have

$$\pi_1^* A_p^1 x_p + \pi_2^* A_p^2 x_p \;=\; 0. \tag{4.2}$$

Given the constraints in the first stage (SG1), i.e., $A_p^1 x_p + A_q^1 x_q \leq b^1$, we have that

$$\pi_1^*(A_p^1 x_p + A_q^1 x_q) \;\leq\; \pi_1^* \cdot b^1 \tag{4.3}$$

becasue $\pi_1^* \geq 0$ as mentioned above.

By (4.1) $-$ (4.2) $+$ (4.3), we can derive

$$\pi_1^* A_q^1 x_q + \pi_2^* A_q^2 x_q \;\leq\; \pi_1^* b^1 + \pi_2^* b^2,$$

which is just the Bender's feasibility cut $\pi_1^* b^1 + \pi_2^* b^2 - \rho^* x_q \geq 0$, since $\rho^* = \pi_1^* A_q^1 + \pi_2^* A_q^2$ according to (D-SG2).

Therefore, we conclude that the cuts generated by our approach, when introduced to the first stage, help form a tighter formulation than the conventional Bender's feasibility cut does. $\square$

Remark 1. In general, for the two-stage stochastic program, the IIS specified by a dual extreme ray can imply the corresponding Bender's feasibility cut (which can be proved by a similar logic as above). However, if we are directly introducing those IIS cuts to the first stage, then additional decision variables (i.e., the second stage variables, y) will be explicitly formulated in the first stage as well, which is typically not computationally preferable or even affordable. In contrast, our approach will only focus on a small portion of the dual values, and then accordingly add just a small subset of the identified IIS to the first stage, while transforming everything into the original variables. Therefore, only a small number of cuts will be introduced to the first stage, and all of them are solely restricted on the original first stage decision variables. Moreover, according to the above theorem and proof, the cuts added to the first stage by our approach are still stronger than the Bender's feasibility cut.

Cut Generation Case 2 — Feasible LP-relaxation

We now consider the case where (R-SG2) is feasible (As we discuss in Appendix 4.7.3, this is theoretically possible, but a rare occurrence in practice). In this case, we can no longer leverage the duality theorem like the previous one to identify a (irreducible) subset of the unenforced constraints which "cause" the infeasibility of the second stage. Instead, we have to potentially enumerate all of those unenforced constraints (i.e., those in $A^2x \leq b^2$) in order to achieve a similar result.

Besides effectively pruning off the current unacceptable partial fixing, there are two aspects we need to take care of simultaneously. On the one hand, the number of the identified constraints must be kept small, because otherwise the first stage problem will become too hard to solve. On the other hand, the identification needs to be completed in a reasonable time, and should never congest the proposed two-stage iterative process. Simply applying the built-in conflict analysis tool of a commercial solver like CPLEX or a well-known filter algorithm, e.g., the additive or deletion method (Chinneck and Dravnieks, 1991; Tamiz et al., 1996), to identify an IIS to our infeasible second stage problem can be very time consuming. This is because there are typically a few thousand unenforced constraints that will potentially be enumerated one by one during the identification process, while, for each iteration, a large-scale IP needs to be solved. Thus, in order to address these two issues and balance the size and the time of the identification, we propose a new cut generation (identification) approach based on a customized filter procedure, similar to the additive/deletion hybrid method along with the grouping strategy by Guieu and Chinneck (1999), to efficiently deal with this infeasibility case, which is detailed as follows.

We first perform an additive procedure to the subsystem of the second stage model (SG2), where the constraints that are not enforced in the first stage, $A^2y \leq b^2$, are tentatively removed (i.e., the following F with $K = \mathbb{R}^n$). However, rather than enumerating each unenforced constraint individually, we each time consider restoring a group of those unenforced constraints, which are related to a specific service, back to this subsystem. More specifically, we loop through each service (more precisely, each of those not in L), and add the corresponding unenforced constraints of it (denoted as D), i.e., imposing some coverage requirement on this specific service, to polyhedron K (initialized as $\mathbb{R}^n$). We repeat updating/tightening K through $K \leftarrow K \bigcap D$ as we go through all services, until the region specified by the following formulation F becomes empty.

$$F := \{y \in K \mid A_p^1 y_p + A_q^1 y_q \le b^1,\ x_q^* - y_q \le 0,\ y \text{ integer}\}.$$

If the number of constraints in K is small enough (e.g., less than 5% of the total number of unenforced constraints), then we bring them all back to the first stage, and accordingly update the partitioning of the original linear system $Ax \le b$. Otherwise, we further apply a conventional IIS filter algorithm like the deletion approach (or a solver's built-in conflict analysis tool) to K, to further reduce it to an irreducible subset, which still maintains F to be empty. Then, as before, we add this irreducible subset of constraints to the first stage, and update the partition accordingly. Algorithm 5 below presents this filter procedure, using the deletion approach, in more detail (line 10 – 15), while also providing the pseudocode of the whole cut generation approach we propose here for handling this case (i.e., (R-SG2) is feasible).

Note that only a small number of services will typically be looped through before F becomes empty. This means that we need to solve a very small number of IPs during this step, while the resulting K will be much smaller than the original, entire set of unenforced constraints, for the application of the follow-up filter procedure (i.e., the deletion procedure), if necessary. Therefore, the whole process can be completed in a much shorter time than directly using a conventional approach or the built-in tool of a solver, although the constraints we bring back to the first stage are no longer always irreducible. In other

Algorithm 4

```
1: Initialize $K = \mathbb{R}^n$;
2: for each service $s$ not in $L$ do
3:     Let $D_s$ be (the polyhedron defined by) the set of unenforced constraints among $A^2x \leq b^2$, which
       each corresponds to a specific coverage requirement imposed on service $s$;
4:     $K \leftarrow K \bigcap D_s$;
5:     if region $F := \{y \in K \mid A^1_p y_p + A^1_q y_q \leq b^1,\ x^*_q - y_q \leq 0,\ y \text{ integer}\}$ is empty then
6:         break;
7: if the number of constraints in $K$ is greater than 5% of the current unenforced ones then
8:     for each constraint $d \in K$ do
9:         $K \leftarrow K \backslash \{d\}$, i.e., remove constraint $d$ from polyhedron $K$
10:        if region $F \neq \emptyset$ then
11:            $K \leftarrow K \bigcap \{d\}$, i.e., bring constriant $d$ back to $K$
12: Incorporate the constraints in $K$ to the first stage formulation (SG1), and accordingly update the parti-
    tioning of the original linear system:
        $\{A^1x \leq b^1\} \leftarrow \{A^1x \leq b^1\} + K \qquad \{A^2x \leq b^2\} \leftarrow \{A^2x \leq b^2\} \backslash K$
```

words, our proposed approach also trades off between the size of the generated cuts and the generation time, which can overall provide a more desirable performance.

Remark 2. In practice, we may apply this proposed service-level additive procedure (line 2 – 8 in Algorithm 5) to Case 1 as well, if the corresponding LP-relaxation (R-SG2) is too hard to solve (i.e., prove infeasibility). More specifically, we terminate the dual simplex algorithm once a specific time limit is reached, and apply our proposed additive procedure to obtain a much smaller infeasible subsystem (i.e., region F when the for-loop in line 2 is broken). Then, we start things over, and apply the dual simplex algorithm to solve the LP-relaxation of this identified subsystem instead. We follow the steps described in Section 4.4.3 to generate cuts if this relaxation turns out to be infeasible too. Otherwise, we resume the rest of the steps in Algorithm 5, and may further reduce the identified constraints to an irreducible subset through a filter procedure.

4.4.4 A Network-based Model for Service Selection

A key decision required by our proposed two-stage partial fixing approach is to select the services (i.e., the set L) whose assignments are to be partially locked in after the first phase. That is, for these services we will lock in all the decisions that ensure their corresponding coverage constraints are satisfied and fix these as inputs to the second stage. The selection of these services L will significantly impact the computational performance of the whole two-stage solution framework, because it is directly related to the possibility that we will encounter infeasibility during the second phase. On the one hand, if a service has limited flexibility (e.g., tightly-bounded coverage constraints), then there will be little room to make additional assignments for it in other unlocked cells in the scheduling table (Figure 4.1) in the second stage, which might make it difficult to satisfy the coverage requirements of the remaining, non-selected services. On the other hand, flexibility is also frequently limited by the *interaction between groups of services* that collectively are very restrictive with respect to a set of potential assignments, and thus should be collectively included in the set L.

More specifically, our service selection method is designed mainly based on the following two observations:

- First, we observe that it would be preferable to select a service which has wide bounds on its coverage requirements (i.e., a big difference between the corresponding upper bound and lower bound on the number of residents needed for the specified block(s)). The reason is that, typically, as few assignments on this service (if selected in set L) as possible will be made in the first stage because of the design of the objective function. Therefore, after the assignments specified by the first stage's solution have been locked in, it's very likely that there is sufficient room left for us to make additional assignments on this service, to ensure the remaining coverage constraints for those non-selected services can be satisfied during the second stage.

For example, suppose that the general emergency room service (ER) requires exactly two residents during each block, while the pediatric emergency room service (PER) requires 1 or 2 resident during each block. Suppose also that there are three residents who have to be assigned to either ER or PER in a given block, according to their respective educational requirements. In this case, suppose that service ER is included in set *L*, i.e., selected to be partially fixed (assuming PER is not). We may find that in the first stage none of these three residents was assigned to service ER during that block (i.e., two other residents will be assigned to ER to ensure the corresponding coverage requirement is satisfied), since assigning all of them to service PER could be feasible because service PER is not included in *L*. However, in doing so, this will violate the coverage constraints for service PER by assigning too many residents during the second stage. Because service ER has limited flexibility (i.e., the required 2 slots of coverage has been locked in for those two other residents), we cannot move any of those three residents from service PER to service ER in the second stage and therefore the second stage will be infeasible.

- We also observe that if two services compete heavily for a shared set of resources (i.e., a specific group of residents over a specific set of time periods), then it is likely for infeasibility to arise during the second stage if only one of them is included in set *L*. In other words, it would be preferable to bind two specific services when forming set *L*, if there are coverage requirements on them collectively specifying a relatively high lower bound on the number of required assignments (i.e., the number of residents assigned during the specified time periods).

  For example, assume there are, in total 10, residents in a specific cohort. Suppose the general medicine service (GM) requires at least 5 but no more than 7 of them to provide coverage during the second half of October, while the generalist service (General) also needs at least three residents from this cohort during this block. Suppose also that three specific residents in this cohort, denoted as Resident-1, Resident-2 and Resident-3, who are all required to do 2 blocks of service in either AMB, VW or GM during the whole October, according to their educational requirements. If we only select service GM to form set *L* without service General, then it is possible that during the first stage, we decide to have Resident-1 to perform service AMB, Resident-2 and Resident-3 to do service VW during the whole October, while selecting some other 5 residents from the cohort to provide coverage to service GM. Since these 5 assignments on service GM will be locked in the second stage, while Reisident-1, Resident-2 and Resident-3 cannot provided coverage for service General during the second half of October, then we will be short of residents for covering service General when its coverage requirement is factored in, and thus the second stage will be infeasible.

To incorporate these two aspects into the service selection, our idea is to use an undirected network to link different services, where each node (i.e., service) carries a flexibility measurement value (for the first bullet above) while a weight value is designed and assigned to each edge to represent how intensively the corresponding two services compete for assignments (for the second bullet above). More specifically, the smaller the node flexibility measurement value is, the more likely that selecting the corresponding service will cause infeasibility during the second stage; the greater the weight for an edge, the more urgent it would be to bind the selection of the corresponding two services. Then, for each service (node), we normalize the weights of its associated edges into "probability" values, which each can be interpreted as how "likely" we will encounter infeasibility during the second stage if we select only this service without selecting the other one specified by the corresponding edge as well. Based on this design, in addition to the normalized flexibility measurement values across all services, we derive a *pickiness score* for each service to reflect the necessity of selecting this specific service, in order to avoid infeasibility, through solving a system of linear equations. Based on these scores, we lastly select a handful of services among the top to constitute set L. A more detailed description of each of these steps is provided as follows.

We construct an undirected graph $G(S,E)$, where the set of nodes in the network is the set of services S. Each node (service) s has a value h_s, which reflects the flexibility of making additional assignments on service s if we choose to partially fix it based on first stage's solution. Each edge $e := (i,j) \in E$, connecting services (nodes) i and j, is associated with a weight w_e to indicate how intensively these two services compete for the constrained resource. The complete description and logic of the calculation of values h_s and weights w_e are provided below. Figure 4.3 provides a simple visualization of this proposed network.

Let Cov be the set of service coverage requirements (i.e., Group 3 in Section 4.3). For each $c \in Cov$, denote its involved set of residents/services/time periods, respectively, as $c_R/c_S/c_T$, while its lower/upper bound value as c_{lb}/c_{ub}. Then, for every node (service) $s \in S$, we choose to estimate its flexibility h_s, defined as:

$$\text{(flx)} \qquad h_s = \sum_t \left\{ \min_{\substack{c \in Cov: \\ c_S = \{s\},\ c_T = \{t\}}} \frac{c_{ub} - c_{lb} + 0.1}{|c_R| - c_{lb}} \right\} \qquad \forall s \in S.$$

A higher value of h_s represents higher flexibility of the corresponding service in terms of the bounds of its coverage requirements, which indicates that service s is a good choice to be included in the selection L for the first stage.

To calculate h_s, we loop through each block t in the planning horizon. For each block,

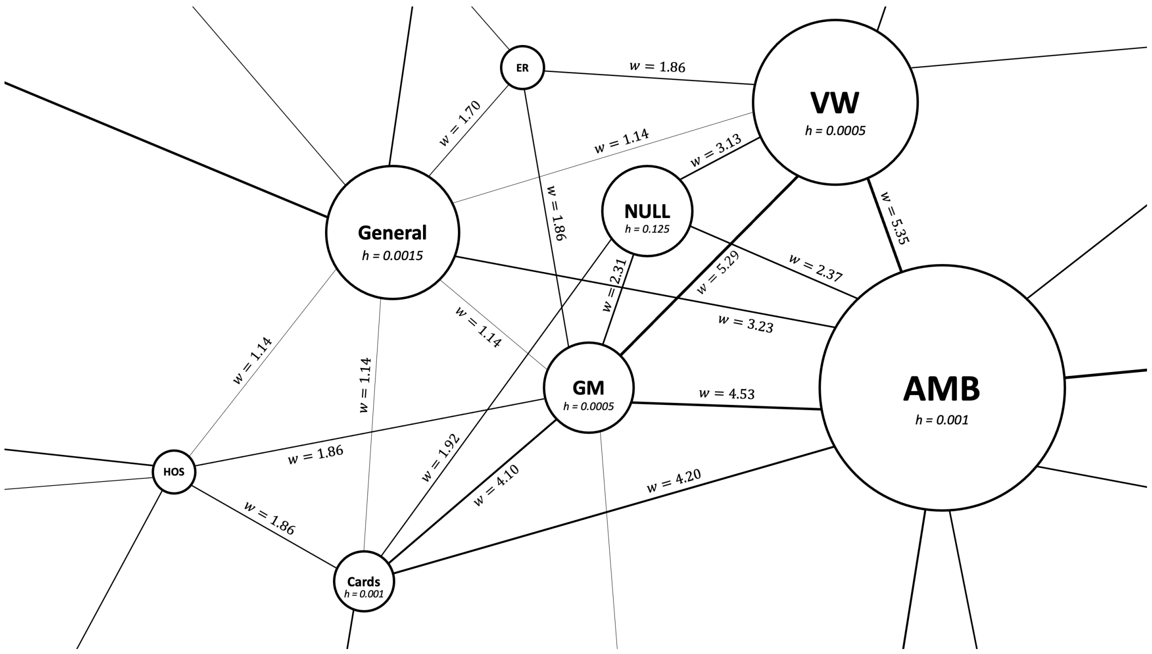

Figure 4.3: A visualization of the proposed network for the service selection. The size of each node reflects the value of its pickiness score k obtained by solving the proposed linear equations system (PR).

we find the *tightest* coverage requirement c which imposes restriction solely on service s during this specific block (i.e., $c_S = \{s\}$ and $c_T = \{t\}$), and add the ratio between the gap of its lower bound and upper bound (i.e., $c_{ub} - c_{lb}$) and the excessive number of involved residents, compared to its minimum required headcount (i.e., $|c_R - c_{lb}|$), to the flexibility measurement value h_s. Note that it is the most common case for practical instances that a coverage requirement is simply on a single service while on a single block.

Note that the value of h_s increases when all constraints on service s have significant range between the upper and lower bound, i.e., when there is the option to assign it to additional residents in the second stage if needed. Additionally, we choose to divide the range by the excessive number of residents, instead of simply using the range value for the calculation of h_s, in order to differentiate coverage requirements that have the same difference between the respective lower bound and upper bound. More specifically, if the opportunity to be additionally assigned to service s in the second stage needs to be shared by a larger group of people, then it indeed offers less flexibility. This is because the larger the number of surplus residents, the more likely the assignments on service s that turn out to be additionally required in the second stage will be a large number, and thus the more likely we will run out of the buffer offered by the range of the coverage requirement in the second stage. From another angle, if a coverage constraint has very narrow bounds (e.g., $c_{ub} = c_{lb}$), then the larger the specified group of residents means the more people are subject to this restriction (in other words, the fewer people, out of this group, are free from the control of this restriction, who can be assigned to service s arbitrarily in the second stage if needed). Lastly, the 0.1 in the numerator of (flx) is added to ensure that when multiple requirements each respectively have a zero gap (i.e., $c_{ub} = c_{lb}$), i.e., offer no flexibility on making additional assignments during the second stage on service s at some block t, then the one which has more surplus residents over the required minimum headcount will be considered to have tighter coverage requirement for the calculation of h_s at t, as its tightness will impact a larger number of residents.

For each edge $e \in E$ connecting service i and j, we define its weight by the following expression:

$$\text{(wgt)} \qquad w_e := \sum_{\substack{c^1,c^2 \in Cov:\ c_S^1=\{i\},\\ c_S^2=\{j\},\ c_T^1=c_T^2,\\ c_R^1 \subseteq c_R^2 \text{ or } c_R^2 \subseteq c_R^1}} \frac{c_{lb}^1 + c_{lb}^2}{|c_T^1| \cdot \max\{|c_R^1|,\ |c_R^2|\}} + \sum_{\substack{c \in Cov:\\ i \in c_S,\ j \in c_S}} \frac{c_{lb}}{|c_R| \cdot |c_T|} \cdot \frac{2}{|c_S|}.$$

A higher value of w_e represents higher intensity of the competition between the corresponding two services for the shared sets of resident resources, which indicates that the selection of the two services that edge e connects should be bounden when forming set L for the first stage.

The idea underlying this calculation design is that, if services i and j need to coordinate their assignments on some set of cells in the scheduling table (Figure 4.1), then the corresponding weight w_e will increase. The more challenging this coordination is (i.e., the more intensive the competition between them for the involved residents across the specified blocks there is), the larger w_e will grow. More specifically, for the first term in (wgt), we consider the case that services i and j compete with each other through two separate coverage requirements c^1 and c^2, where one is imposed solely on service i (i.e., $c^1_S = \{i\}$) while the other one is solely on service j (i.e., $c^2_S = \{j\}$). If the time period sets specified by these two coverage requirements are exactly the same (i.e., $c^1_T = c^2_T$), and if there exists an inclusive relationship between their resident sets (i.e., $c^1_R \subseteq c^2_R$ or $c^2_R \subseteq c^1_R$), then we know that services i and j will compete across, in total, $|c^1_T| \cdot \max\{|c^1_R|,\ |c^2_R|\}$ cells for at least $c^1_{lb} + c^2_{lb}$ assignments. Therefore, we add the corresponding ratio (which can be interpreted as the minimum amount of contribution required by each involved cell for the satisfaction of these two requirements, if evenly distributed) to the weight of edge (i, j) to reflect how tight the room is to satisfy the assignments of these two services. The same logic is applied to the second term in (wgt). The only difference is that in this case we focus on each single coverage requirement c which imposes restrictions on multiple services together, where services i and j are included simultaneously (i.e., $i, j \in c_S$). Again, the value of this term represents the evenly-distributed contribution required from each involved resident during each involved block to the assignments on these two services i and j by the coverage requirement c. Since this per-cell contribution also reflects the intensity of the competition among services i and j within a specific set of cells in the scheduling table, we add it to weight w_e as well.

Given this graph G with flexibility measurement values $h \in \mathbb{R}^{|V|}$ and weights $W \in \mathbb{R}^{|V|\times|V|}$ defined above, we propose to use the following linear system to formulate the pickiness scores $k \in \mathbb{R}^{|V|}$, which can be viewed as a variant of the PageRank model (Brin and Page, 1998).

$$\text{(PR)} \qquad k \ = \ \theta \bar{W} k + (1-\theta)\bar{h}.$$

Here, $\bar{W} := \{\bar{w}_{i,j}\}$ is a stochastic matrix, which is derived by normalizing matrix W

by column (i.e., $\sum_i \bar{w}_{i,j} = 1$ for every j). $\bar{w}_{i,j}$ represents the relative necessity of selecting service i, in order to avoid infeasibility during the second stage, if service j has already been selected. Similarly, vector $\bar{h}$ is the normalization of h, which reflects the relative likelihood that the second stage will turn out to be feasible by considering the flexibility of each service independently. Parameter θ, a scalar in $(0,1)$, specifies how the competitive relationships among different services and the flexibility offered by each individual service are balanced in calculating the pickiness scores k.

By this design, on the one hand, the larger the h_s is for a specific service s (which means the more flexibility there is for making additional assignments on s during the second stage), the larger $\bar{h}_s$ will be after normalization, and therefore the larger the resulting pickiness score k_s could be, which means the more likely service s will be selected, as desired. On the other hand, a large value of w_e for a specific arc $e := (i,j)$ (which means services i and j compete heavily with each other) is more likely to result in a large value of $\bar{w}_{i,j}$ and $\bar{w}_{j,i}$. Therefore, if eventually service i carries a big pickiness score k_i, then service j is likely to take a big pickiness score k_j as well (and vice versa), which implies a binding relationship between these two services with respect to the selection for set L, as desired. Moreover, this design implicitly ensures that service s which has very low demand on residents across all blocks (i.e., small lower bound values for all involved coverage requirements) will get a small pickiness score k_s because $\bar{w}_{s,i}$ for all $i \in S$ should be very small, as service s then barely competes with any other service for any shared resources. This implication is also desired because we want to avoid selecting a service which cannot result in a sufficient number of assignments locked in, in order to ensure the problem size of the resulting second stage will be significantly reduced. In summary, by the design of the flexibility measurement values h, the weights W, and the linear equations system (PR), the two aspects mentioned at the beginning of this sub-section are incorporated simultaneously, which is thus expected to help us determine a high-quality service selection for initiating our proposed two-stage partial fixing approach.

By Perron–Frobenius theorem, we know that matrix $I - \theta\bar{W}$ is non-singular for arbitrary $0 < \theta < 1$ (assuming that the proposed network graph $G(S,E)$ as shown in Figure 4.3 is strongly connected with strictly-positive-weight edges; this assumption should generally hold for any real-world instance, given the way the weights are derived, i.e., (wgt), and the structure of the coverage requirements on all of the services in practice). Therefore, according to (PR), our pickiness scores $k = (1-\theta)\cdot(I-\theta\bar{W})^{-1}\bar{h}$. In practice, we use a power iteration method as in Arasu et al. (2002) to instead approximate k by convergence (a copy of the pseudo-code is provided in Appendix 4.7.4 for reference), which is terminated when the norm of the difference between two subsequent iterations is within a specific small threshold. Once k is obtained, we form set L for our proposed two-stage partial fixing solution framework by selecting a handful of services with the highest pickiness scores indicated by k.

Note that the linear equations system (PR) we propose here is not the only possible way to help us determine set L to achieve a desirable performance of the two-stage framework. For example, selecting services simply based on the cumulative weights across all edges for each node in the network may also be sufficient to ensure an adequate computational performance. For the same reason, there exist many other ways to define the weights W and the flexibility values h, or even other ways to design the network structure, which also make sense and are effective. We choose to only propose this specific approach here, since our primary goal is simply to ensure a good service selection, while there is no need (or it's yet impossible) to identify the "best" one. In addition, our experiments (see the next section) have demonstrated that this proposed approach works sufficiently well for all instances during the past years at UMMS.

4.5 Computational Experiments

We implemented our proposed approach using C++ (Visual Studio 2017) with CPLEX (version 12.80) on a 64-bit operating system computer with two 2.10GHz processors and 128GB RAM. The CPLEX solver was configured with its default settings for all experiments in this section, unless otherwise specified. To evaluate the effectiveness of our approach, we applied it to solve three real-world instances from our clinic collaborator UMMS for constructing its resident annual block schedules for the past three years, and we present the results in this section.

Table 4.2 summarizes the basic information of each of these three instances, and the computational performance of directly solving the corresponding base model (IP) by CPLEX. More specifically, the second, third, and fourth column, respectively, provide the number of residents, services, and blocks (i.e., time periods) considered in the corresponding instance. The table's fifth and sixth column present the number of rows and columns (i.e., constraints and variables) that the base model (IP), i.e., the integer program formulation, consists of, while the the seventh and eighth column show the number of constraints in the base model for enforcing the resident education requirements and the service coverage requirements (i.e., Groups 2 and 3 in Section 4.3). The last column gives the total runtime of CPLEX to solve the corresponding base model.

Table 4.2: Basic information of the real-world instances, and the performance of a conventional approach, i.e., solving the base model (IP) by CPLEX.

Instance	#Resdt	#Serv	#Blk	#Row	#Coln	#Edu	#Cov	Runtime
Year-2018	249	92	24	1.8M	1.2M	8,693	3,398	32 mins
Year-2019	250	95	26	2.7M	1.3M	26,498	4,205	272 hrs
Year-2020	253	94	26	2.4M	1.3M	23,651	3,972	236 hrs

Note that the numbers in the table suggest that the constraints for education and coverage requirements are only a small fraction of the total number. In fact, this is because many of the constraints in the model are to ensure the structure of the output schedule, while there are some miscellaneous rules (i.e., Group 4 in Section 4.3) that each require a series of constraints, rather than just a single constraint, to be defined in the model to enforce. However, this does not mean the resident education requirements and the service coverage requirements only comprise a tiny portion of all the explicit rules a valid resident block schedule should satisfy. We refer readers to Appendix 4.7.1 to see the complete requirements definition and the full formulation to get a better sense of this point.

We can observe that, even for the middle-size instance Year-2018, where there are much fewer educational requirements and other miscellaneous rules, the base model will have a huge size, and it will take a significant amount of time for CPLEX to solve. Furthermore, the runtime for the large-scale instances, Year-2019 and Year-2020, will explode to be more than 1 week, which is unacceptable for practical implementation. These results require us to develop new approaches to overcome the computational issue and more efficiently complete the resident schedule construction, such as the one we've presented in the previous sections.

We present the results of the experiments on applying our approach to the three instances, respectively, in Table 4.3, 4.4, and 4.5 below. For each experiment, we first construct and solve the network-based model proposed in Section 4.4.4, where we set balance parameter $\theta = 0.85$ in formulation (PR), and at the end select six services that have the hightest pickiness score according to the converged k values. Please note that based on our experiments, the eventual service selection was shown to not be sensitive to the value of θ (by varying it from 0.8 to 0.95). Once this process is done, we initiate the two-stage partial fixing approach described in Section 4.4.3 with the selected services to identify a feasible block schedule to the corresponding instance. Here, we set the time limit to be 1 hour and the stopping optimality gap to be 5% for solving the first stage model (SG1) for instance Year-2019 and Year-2020. This setting ensures we provide a sufficient amount of time to the solver to optimize the provided objective function for finding a high-quality partial fixing on the selected services, but prevent it from spending an excessive amount of time on it and ruining the overall performance. We decrease these two parameters to 5 minutes and 1%, respectively, for instance Year-2018, because we know that it is much smaller and easier to solve, and are confident that a near-optimal solution for its first stage formulation can be obtained within a short time.

Table 4.3: Results of applying the proposed two-stage partial fixing approach to instance Year-2018.

TotTime	#Itr	#ICstr	#FCstr	Iteration Details					
				#Assign	Additive	Dual	Filter	SecTime	#RCstr
6 mins	2	435	444	1,488	No	Yes	No	28 secs	9
				1,488	-	-	-	41 secs	-

Table 4.4: Results of applying the proposed two-stage partial fixing approach to instance Year-2019.

TotTime	#Itr	#ICstr	#FCstr	Iteration Details					
				#Assign	Additive	Dual	Filter	SecTime	#RCstr
1 hr	2	562	588	1,738	No	Yes	No	30 secs	26
				1,739	-	-	-	27 mins	-

Table 4.5: Results of applying the proposed two-stage partial fixing approach to instance Year-2020.

TotTime	#Itr	#ICstr	#FCstr	Iteration Details					
				#Assign	Additive	Dual	Filter	SecTime	#RCstr
47 hrs	5	566	722	1,687	Yes	No	Yes	49 mins	52
				1,687	Yes	No	Yes	10 hrs	52
				1,695	No	Yes	No	2 mins	26
				1,695	No	Yes	No	8 mins	26
				1,689	-	-	-	29 hrs	-

In these tables, the first column shows the total runtime for the proposed two-stage approach to solve the corresponding instance, while the second column indicates the number of iterations it takes. The third and fourth column present the number of coverage constraints incorporated in the first stage's model (SG1) during the first and last iteration respectively, and thus their difference corresponds to the number of coverage constraints restored by our cut generation approaches during the entire process. The remaining columns in the table provide detailed information for each specific iteration. More specifically, the fifth column shows the total number of assignments that are fixed for the selected services (i.e., whose corresponding value in the first stage's solution x_q^* is 1) during the second stage. Column 'Additive' indicates whether the proposed service-level additive procedure in Section 4.4.3 is triggered or not. In other words, according to the practical implementation mentioned in **Remark 2**, it will show 'Yes' if the dual simplex algorithm fails to conclude whether the LP-relaxation of the second stage's model is feasible within the given time limit (which we set to be 1 hour here); and show 'No' otherwise. Column 'Dual' indicates whether the restored constraints are identified based the dual extreme ray by the dual simplex algorithm as described in Section 4.4.3. Column 'Filter' indicates whether the filter procedure will be applied to further reduce the size of the identified constraint set (it is 'Yes' if and only if dual simplex algorithm fails to specify an IIS, while the resulting set K by the additive procedure is not small enough). The second-to-last column shows the solution time for solving the second stage model if it is the last iteration (i.e., a feasible schedule is then constructed and the entire process terminates). Otherwise, it presents the total runtime cost by the cut generation approaches to identify the unenforced constraints for eliminating the current infeasible partial fixing (in this case, the infeasibility of the second stage's model can typically be proved in a very short time, so we choose to not show that in this table due to the space limit). The last column presents the number of unenforced constraints that have been identified, and are to be restored in the first stage's formulation for the next iteration.

By these results, our approach is shown to be able to solve the resident block scheduling problem in a much shorter time than the conventional approach that applies the mixed integer program (MIP) techniques directly. Specifically, a feasible solution to Instance

Year-2018 and Year-2020 can be identified more than 5 times faster, while the total runtime is decreased from 272 hours to only 1 hour for solving Instance Year-2019. Moreover, the effectiveness of our approach is demonstrated by the small number of iterations it takes as well as the very limited portion of the coverage constraints that need to be incorporated for producing a feasible schedule.

There are a few critical aspects of our proposed solution framework remaining for evaluation. First, fixing the number of selected services to be six may not be a good choice for all cases. For example, the very long runtime, i.e., 29 hours, for solving the final second stage model for Instance Year-2020 as shown in Table 4.5 suggests that more services would preferably be selected to further alleviate the challenges of this solution process. Moreover, we need to make sure we do not make the conclusion on the performance of our approach based on either the best or the worst scenario in terms of the number of services selected. Thus, we want to further perform an evaluation by varying this number for all instances. Second, we want to overall justify whether it is beneficial to build a complicated network-based model to assist us with the service selection. In addition, the effectiveness of the proposed cut generation mechanisms as well as the incorporation of a well-designed objective function in the first stage should be verified. We've carried out experiments to accordingly evaluate each of these aspects, and the results are presented in the next sub-sections.

4.5.1 Experiments on the Number of Selected Services

We vary the number of the top pickiness score of the services that we select to initiate the two-stage partial fixing approach for all three instances. The results are provided in the following Tables 4.6, 4.7, and 4.8. Note that unlike in the previous tables, the detailed information for each iteration of the two-stage process is not provided here due to the limited space. Instead, the last two columns, '#FAssign' and 'FSecTime', respectively, present the number of assignments fixed for the selected services in the second stage for the final iteration and the time for solving this final second stage's formulation to obtain the feasible schedule.

Table 4.6: Results of applying the proposed approach with varying number of selected services to instance Year-2018.

#Serv	TotTime	#Itr	#ICstr	#FCstr	#FAssign	FSecTime
3	20 mins	1	289	289	758	19 mins
4	6 mins	1	339	339	1,046	4 mins
$\underline{5}$	$\underline{\text{3 mins}}$	1	387	387	1,278	1 mins
6	6 mins	2	435	444	1,488	41 secs
7	19 mins	3	483	498	1,693	23 secs

Table 4.7: Results of applying the proposed approach with varying number of selected services to instance Year-2019.

#Serv	TotTime	#Itr	#ICstr	#FCstr	#FAssign	FSecTime
4	73 hrs	2	404	430	1,252	73 hrs
5	13 hrs	3	484	784	1,711	3 hrs
$\underline{6}$	$\underline{\text{1 hr}}$	2	562	588	1,739	27 mins
7	36 hrs	10	608	2,142	2,185	19 mins

Table 4.8: Results of applying the proposed approach with varying number of selected services to instance Year-2020.

#Serv	TotTime	#Itr	#ICstr	#FCstr	#FAssign	FSecTime
6	47 hrs	5	566	722	1,689	29 hrs
$\underline{7}$	$\underline{\text{5 hrs}}$	3	649	753	1,803	17 mins
8	6 hrs	4	727	831	2,061	14 mins
9	11 hrs	5	783	918	2,215	46 mins

By these experiments, we see that a better overall computational performance can indeed be achieved by selecting a different number of services with the top pickiness scores for different instances. More specifically, we can observe that, for the most part, the trend is that the more services we select the less time will be spent by solving the final second stage formulation, but the more iterations may be needed to reach that final step. This matches the intuition naturally, as the more assignments we fix, the smaller size the second stage problem can be reduced to, while it is more likely that we enforce part of those assignments at some wrong spots, which then causes infeasibility for completing the rest of the puzzles. In other words, the size of the selected services trades off the time spent on solving the final second stage formulation and the time spent on cut generation for correcting infeasible assignment fixings. As we can observe a V-shape in terms of the total runtime as we vary the number of selected services, it would be beneficial to be able to identify the “sweet spot” up front for each specific instance. However, this aspect is currently not considered in this research, and we plan to explore how to achieve that as future work. Lastly, we want to point out that although the computational performance varies significantly depending on the size of the service selection, we can still always largely reduce the total runtime for solving the resident block scheduling problem compared to using the traditional MIP techniques, regardless of the number of services we select.

4.5.2 Effectiveness of the Network-Based Model for Service Selection

The objective of the proposed network-based model in Section 4.4.4 is to assist us with a high-quality service selection (i.e., whose assignments should be partially locked in during the second stage), so that the proposed two-stage approach can overall achieve an efficient and robust performance. To evaluate whether this objective can be achieved adequately by the proposed design, we compare the computational performance of the service selection based on our network-based model against two other selection methods, using Instance Year-2019. The results are summarized in Table 4.9 below. Here, for each fixed selection size (from four to seven), we present the results for the total of three service selection methods, specified by Column ‘SlctMtd’, in three rows. Row ‘NTW’ stands for the method of selecting the services with the top pickiness scores k by solving (PR) in the network-based model we propose; Row ‘TEN’ is a random selection among the ten services who have the greatest pickiness scores by our network-based model; 'RDM' corresponds to a random selection among all services involved in the instance.

Table 4.9: Comparisons on the computational performance achieved by three different service selection methods.

#Serv	SlctMtd	TotTime	#Itr	#ICstr	#FCstr	#FAssign	FSecTime
4	NTW	73 hrs	2	404	430	1,252	73 hrs
	TEN	50 hrs	2	342	461	936	44 hrs
	RDM	> 3 D	3	263	524	733	-
5	NTW	13 hrs	3	484	784	1,711	3 hrs
	TEN	15 hrs	1	508	508	1,152	15 hrs
	RDM	> 3 D	6	542	1,295	842	-
6	NTW	1 hr	2	562	588	1,739	27 mins
	TEN	57 hrs	6	591	1,169	1,730	33 hrs
	RDM	63 hrs	8	384	834	1,364	39 hrs
7	NTW	36 hrs	10	608	2,142	2,185	19 mins
	TEN	28 hrs	11	687	1,823	1,968	25 mins
	RDM	52 hrs	2	381	382	1,094	51 hrs

We can observe that methods 'NTW' and 'TEN' can almost provide a comparable result, except for the case of selecting six services where the 'NTW' method results in a much better performance. On the other hand, method 'RDM' consistently achieves the worst result, and there are even a couple of times that it fails to make the two-stage iterative partial fixing process converge within the time limit, i.e., 3 days. Therefore, we can conclude that the proposed network-based method is effective and robust for clustering preferable services for partial fixing, and it can make those services stand out more likely.

4.5.3 Effectiveness of the Cut Generation Mechanisms and the Design of the First Stage's Objective Function

Table 4.10 reflects the likelihood that the second stage's model is feasible based on the final optimality gap when the solution process of the first-stage problem terminates. More specifically, the first row in this table displays the frequencies of achieving the corresponding optimality gap, as indicated by the column, when the first stage is solved; the second row shows the number of times among those occurrences that the solution to the first stage specifies a feasible partial fixing and thus a feasible schedule can be obtained by solving the subsequent second stage. Please note that the information presented here not only includes the results from the iterations in the previous experiments that have been shown above, but also the results of some other preliminary/intermediate experiments, e.g., using various time limits on the first stage's solution process, on different random service selections, and so on (where the majority are on instance Year-2018 because of its smaller size for testing efficiency).

Table 4.10: The relation between the optimality gap achieved for solving the first stage formulation and the feasibility status of the subsequent second stage.

OptGap	$< 5\%$	5-10%	10-20%	20-50%	$> 50\%$
#Occurrence	86	7	3	2	92
#FeasSecStg	62	4	1	2	23

Although a feasible solution to the first stage's model can frequently be obtained within a short time, it is likely ($1 - 23/92 = 75\%$) the subsequent second stage will be infeasible according to this table, if that first stage's solution does not achieve a good objective value, while we cannot further improve it within the rest of the time limit. However, if we are lucky that that arbitrary solution is near optimal, or if we successfully reduce the optimality gap to a small value, then we are almost 3 times more likely to get a feasible solution in the second stage. Moreover, this table also shows that for the majority of the cases we test, the corresponding iterative two-stage process has been terminated by a first stage's solution which is close to optimal. All of these demonstrate that the design of our objective function in the first stage's formulation is effective to reduce the likelihood that we encounter infeasibility when solving the second stage.

Next, we further compare the performance of our proposed approach (denoted as 'PPS') against its two variations with a different configuration, where one implements the no-good cut (No-good) to prune off the infeasible partial fixing on the selected variables (denoted as 'NGC'), instead of the cut generation mechanisms detailed in Section 4.4.3 and 4.4.3, while the other one does not incorporate any objective function $f(x)$ in the first stage (denoted as 'NOF'), and thus will terminate solving it once any feasible solution has been found. The experiment results on the three real-world instances are presented in Table 4.11, 4.12, and 4.13 respectively. To provide a reference, the computational results of solving the base model directly using the conventional MIP techniques by CPLEX are in addition pinned at the top of each of these tables, where the values under Column '#ICstr' and '#FCstr' in the corresponding row 'MIP' indicate the total number of the coverage constraints in the formulation (IP) instead.

Table 4.11: The results of applying the proposed solution approach and its two variations to instance Year-2018.

#Serv	Config	TotTime	#Itr	#ICstr	#FCstr	#FAssign	FSecTime
-	MIP	32 mins	-	3,398	3,398	-	-
3	PPS	20 mins	1	289	289	758	19 mins
	NGC	20 mins	1	289	289	758	19 mins
	NOF	11 mins	1	289	289	762	11 mins
4	PPS	6 mins	1	339	339	1,046	4 mins
	NGC	6 mins	1	339	339	1,046	4 mins
	NOF	33 mins	1	339	339	1,047	32 mins
5	PPS	3 mins	1	387	387	1,278	68 secs
	NGC	3 mins	1	387	387	1,278	69 secs
	NOF	2 mins	1	387	387	1,278	89 secs
6	PPS	6 mins	2	435	444	1,488	41 secs
	NGC	5 mins	2	435	435	1,488	35 secs
	NOF	4 mins	2	435	445	1,491	37 secs
7	PPS	19 mins	3	483	498	1,693	23 secs
	NGC	60 mins	12	483	483	1,691	30 secs
	NOF	7 mins	3	483	500	1,702	30 secs

According to these comparisons, using no-good cut (No-good) to eliminate infeasible assignment fixing on selected services (i.e., 'NGC') cannot provide a consistent computational performance, as expected. Although it can help the two-stage partial fixing approach

Table 4.12: The results of applying the proposed solution approach and its two variations to instance Year-2019.

#Serv	Config	TotTime	#Itr	#ICstr	#FCstr	#FAssign	FSecTime
-	MIP	272 hrs	-	4,205	4,205	-	-
4	PPS	73 hrs	2	404	430	1,252	72 hrs
	NGC	12 hrs	29	404	404	1,252	10 hrs
	NOF	42 hrs	7	404	1,329	1,441	9 hrs
5	PPS	13 hrs	3	484	784	1,711	3 hrs
	NGC	3 hrs	6	484	484	1,508	2 hrs
	NOF	19 hrs	6	484	1,437	1,673	4 hrs
6	PPS	1 hr	2	562	588	1,739	27 mins
	NGC	1 hr	2	562	562	1,739	29 mins
	NOF	31 hrs	9	562	1,905	1,883	43 mins
7	PPS	36 hrs	10	608	2,142	2,185	19 mins
	NGC	2 hrs	2	608	608	2,098	38 mins
	NOF	22 hrs	7	608	1,874	2,278	27 mins

solve instance Year-2019 universally in a shorter time, it fails to make the process converge within the given 3-day limit for any selection size considered for instance Year-2020. This non-robustness prevents this intuitive method from being used in practice, which justifies the necessity of proposing our cut generation mechanisms.

In addition, we can observe that our proposed approach (i.e., 'PPS') results in a comparable performance with the variation 'NOF,' where the objective function is removed from the first stage formulation (SG1), and we cannot conclude which one is better in practice. The fact is that although the design of the objective function has been shown to be effective previously, there are many iterations where we fail to reduce the optimality gap significantly and the final gap remains very large. For those iterations, computational time is "wasted" without providing the desirable benefits of reducing the likelihood of infeasibility of the second stage, and therefore, if the solution process is terminated instead once a feasible solution has been identified, a better computational performance can naturally be achieved as a result. We believe this is the primary reason why in some cases the proposed approach 'PPS' outperforms the no-objective-function variation 'NOF', while in

Table 4.13: The results of applying the proposed solution approach and its two variations to instance Year-2020.

#Serv	Config	TotTime	#Itr	#ICstr	#FCstr	#FAssign	FSecTime
-	MIP	236 hrs	-	3,972	3,972	-	-
6	PPS	47 hrs	5	566	722	1,689	29 hrs
	NGC	> 3 D	71	566	566	1,689	-
	NOF	36 hrs	4	566	670	1,678	32 hrs
7	PPS	5 hrs	3	649	753	1,803	17 mins
	NGC	> 3 D	73	649	649	1,811	-
	NOF	2 hrs	2	649	701	1,805	38 mins
8	PPS	6 hrs	4	727	831	2,061	14 mins
	NGC	> 3 D	77	727	727	2,061	-
	NOF	5 hrs	5	727	859	2,063	2 hrs
9	PPS	11 hrs	5	783	918	2,215	46 mins
	NGC	> 3 D	80	783	783	2,219	-
	NOF	2 hrs	3	783	839	2,217	34 mins

the other cases 'NOF' dominates 'PPS'. In particular, it explains why the 'NOF' performs consistently better than 'PPS' for solving instance Year-2020, since solver logs indicate that the optimality gaps are never improved after the first feasible solution is found across all of the experiments on it.

Based on this analysis, we ideally can mix the usage of the designed objective function with an empty objective in the first stage of the proposed framework, to overall achieve a shorter total solution time. We may also use them alternatively and vary the time limit set on the process of the first stage in an adaptive way. We leave this enhancement for future consideration.

4.6 Conclusion

In this chapter, we consider a resident annual block scheduling for a medical school, where residents from different programs and levels should be assigned to a set of services in the hospital during each time period across the academic year. In addition to the huge problem size, the coordination between residents' individual educational requirements and the coverage requirements for ensuring appropriate staffing levels in all units, along with a large number of other side constraints, make this scheduling task extremely complicated. Conventional approaches including the MIP techniques have been shown to be computationally insufficient to solve practical instances.

We proposed a two-stage partial fixing framework to address the computational challenge. During the first stage, we solve a relaxed variation of the original problem by only incorporating a limited number of coverage requirements. Based on its solution, we partially fix the assignments for a pre-determined set of services, which are selected upfront, and we then try completing the rest of the pieces in the scheduling puzzle during the second stage, to obtain an overall valid resident schedule. We also developed cut generation mechanisms to prune off unacceptable fixings of the assignments of the selected services to start over, if a feasible solution is failed to be produced during the second stage. Lastly, we proposed a network-based model to assist with the selection of services whose assignments should be partially fixed during the proposed two-stage iterative process. Based on our computational experiments, our proposed approach was demonstrated to be both efficient and robust for solving practical instances for our clinical collaborator, and was shown to totally outperform the conventional MIP approach offered by CPLEX.

In terms of future research, we would like to explore additional cut generation approaches to more efficiently prune off the bad assignment fixing on the selected services once infeasibility arises, since this part is currently the biggest bottleneck for further reducing the convergence time of the two-stage iterative process. In addition, we plan to develop an adaptive mechanism for the incorporation of the objective function during the first stage, and/or a dynamic time limit setting, as mentioned previously, so that the unnecessary solution time spent at this stage could potentially be saved. Lastly, we would like to consider and experiment with other service selection approaches for initiating the proposed two-stage partial fixing process, including using a hypergraph representation to explicitly model the competition intensity among more than two services for the linear equations system. In particular, we also plan to incorporate the decision on the number of services to be selected into the selection process. We expect that this way will allow us to systematically and better balance the trade-off between the total number of iterations (and the associated cut generation time) of the proposed solution framework and the time spent on solving the final second stage problem, and overall improve the computational performance of our approach.

4.7 Appendix

4.7.1 An Integer Program Formulation for the Residency Block Scheduling Problem

In order to precisely define our block scheduling problem, we introduce an additional concept, called *rotation*. Basically, a rotation is the combination of a specific service plus an associated duration, in terms of a number of blocks. It says that a resident assigned to such a rotation will perform the corresponding service for a consecutive number of blocks, specified by the associated duration. For example, if a 4-block AMB rotation is scheduled at the beginning of July, then the assigned resident will do AMB service for 4 consecutive time periods, from July to August (assuming each block's length is a half month). By this definition, our residency scheduling can also be interpreted as assigning the residents in all programs to a sequence of rotations for the upcoming academic year. This rotation concept is necessary to define our problem because some rules impose restrictions on the assignment of rotations rather than on individual services. More details can be found in the definitions of all of the rules as follows.

Notation

Sets & Definitions

R : The set of residents.

S : The set of services.

A : The set of possible rotations.

T : The set of time periods (blocks), indexed as $\{0,1,2,...,23\ (\text{or}\ 25)\}$, each of which corresponds to a half month (or two weeks) in the academic year.

Edu : The set of resident education requirements. Each resident education requirement $ed \in Edu$ is defined by a resident $ed_r \in R$, a set of services $ed_S \subseteq S$, a set of time periods $ed_T \subseteq T$, a lower bound ed_{lb}, and an upper bound ed_{ub}. It says that the total number of blocks in the given time period set ed_T, during which the resident ed_r is assigned to services in the given service set ed_S, should be greater than or equal to the given lower bound ed_{lb} but less than or equal to the given upper bound ed_{ub}.

Cov : The set of service coverage requirements. Each service coverage requirement $cv \in Cov$ is defined by a set of residents $cv_R \subseteq R$, a set of services $cv_S \subseteq S$, a set of time periods $cv_T \subseteq T$, a lower bound cv_{lb}, and an upper bound cv_{ub}. It says that the total cumulative number of blocks for which the given set of residents cv_R are assigned to the given set of services cv_S during the time periods in the given set cv_T must be greater than or equal to the given lower bound cv_{lb} but less than or equal to the given upper bound cv_{ub}.

Par : The set of resident pairing rules. Each pairing rule $pr \in Par$ is defined by two groups of residents $pr_{R1}, pr_{R2} \subseteq R$, two services $pr_{s1}, pr_{s2} \in S$, and two blocks $pr_{t1}, pr_{t2} \in T$. It says that the number of residents in the first group pr_{R_1} that are assigned to the given first service pr_{s1} during the given first block pr_{t1} should be equal to the number of residents in the second group pr_{R2} that are assigned to the second service pr_{s2} at pr_{t2}.

Pre : The set of service pre-assignment requirements. Each service pre-assignment $sa \in Pre$ is defined by a specific resident $sa_r \in R$, a service $sa_s \in S$ and a time period $sa_t \in T$. It says that the given resident sa_r must be assigned to service sa_s during time period sa_t.

SPh : The set of service prohibition requirements. Each service prohibition $sp \in SPh$ is defined by a specific resident $sp_r \in R$, a service $sp_s \in S$ and a time period $sp_t \in T$. It says that the given resident sp_r cannot be assigned to service sp_s during time period sp_t.

APh : The set of rotation prohibition requirements. Each rotation prohibition $ap \in APh$ is defined by a specific resident $ap_r \in R$, a rotation $ap_a \in A$ and a time period $ap_t \in T$. It says that the given resident ap_r cannot start a rotation ap_a from time period ap_t.

Spa : The set of spacing rules. Each spacing rule $sc \in Spa$ is defined by a resident $sc_r \in R$, two rotations $sc_{a1}, sc_{a2} \in A$, and a gap value sc_g in terms of the number of blocks. It says that, for resident sc_r, there must be a minimum number of blocks sc_g between the end of any rotation sc_{a1} and the start of any rotation sc_{a2} assigned to him/her.

$Seq:$ The set of sequencing rules. Each sequencing rule $sq \in Seq$ is defined by a resident $sq_r \in R$, a service $sq_s \in S$ and a group of other services $sq_S \subseteq S$. It says that, before resident sq_r can be assigned to service sq_s, he/she must have already been assigned to at least one of the services in the given service set sq_S previously.

Functions

$d(s):$ the set of rotations $d(s) \subseteq A$ that are associated with service $s \in S$. $d(s) \neq \emptyset$ for $\forall s \in S$.

$l(a):$ the length (in terms of the number of blocks) of rotation $a \in A$.

Variables

$X_{r,s,t}:$ Binary variables for $\forall r \in R, \forall s \in S, \forall t \in T$. 1 if resident r will be assigned to service s during time period t; 0 otherwise.

$Y_{r,a,t}:$ Binary variables for $\forall r \in R, \forall a \in A, \forall t \in T$. 1 if resident r will start a rotation a from the beginning of time period t; 0 otherwise.

Formulation

(a1) $\sum_{s \in S} X_{r,s,t} = 1 \qquad \forall r \in R,\ \forall t \in T$

(a2) $X_{r,s,t} = \sum_{a \in d(s)} \sum_{p=\max(0,\ t-l(a)+1)}^{t} Y_{r,a,p} \qquad \forall r \in R,\ s \in S,\ t \in T$

(a3) $ed_{lb} \leqslant \sum_{s \in ed_S} \sum_{t \in ed_T} X_{ed_r,s,t} \leqslant ed_{ub} \qquad \forall ed \in Edu$

$$\text{(a4)}\ \ cv_{lb} \leqslant \sum_{r \in cv_R} \sum_{s \in cv_S} \sum_{t \in cv_T} X_{r,s,t} \leqslant cv_{ub} \qquad \forall cv \in Cov$$

$$\text{(a5)}\ \ \sum_{r \in pr_{R1}} X_{r,pr_{s1},pr_{t1}} = \sum_{r \in pr_{R2}} X_{r,pr_{s2},pr_{t2}} \qquad \forall pr \in Par$$

$$\text{(a6)}\ \ X_{sa_r,sa_s,sa_t} = 1 \qquad \forall sa \in Pre$$

$$\text{(a7)}\ \ X_{sp_r,sp_s,sp_t} = 0 \qquad \forall sp \in SPh$$

$$\text{(a8)}\ \ Y_{ap_r,ap_a,ap_t} = 0 \qquad \forall ap \in APh$$

$$\text{(a9)}\ \ Y_{sc_r,sc_{a1},t} + Y_{sc_r,sc_{a2},p} \leq 1 \qquad \forall sc \in Spa,$$

$$\forall t \in T:\ t \leqslant |T| - 1 - l(sc_{a1}), \qquad \forall p \in T:\ t + l(sc_{a1}) \leq p \leq sc_g + t + l(sc_{a1}) - 1$$

$$\text{(a10)}\ \ X_{sq_r,sq_s,t} \leq \sum_{p=0}^{t-1} \sum_{v \in sq_S} X_{sq_r,v,p} \qquad \forall sq \in Seq,\ \forall t \in T.$$

Constraints (a1) enforce the basic assignment rules (Group 1 in Section 4.3), i.e., each resident must be assigned to exactly one service for each block. Constraints (a2) build the relationship between variables $\{X_{r,s,t}\}$ and variables $\{Y_{r,a,t}\}$, i.e., connect the assignment of rotations with the assignment of services. Constraints (a3) and (a4), respectively, ensure the satisfaction of the resident education requirements and the service coverage requirements that are described in Section 4.3 (i.e., Group 2 and 3). Constraints (a5) enforce the resident pairing rules, while constraints (a6) ensure that all of the pre-assignments are scheduled properly. Constraints (a7) and (a8) enforce the given service prohibitions and rotation prohibitions, respectively. Constraints (a9) enforce the spacing rules, while constraints (a10) ensure all of the sequencing rules are satisfied.

4.7.2 The Design of the Objective Function and Auxiliary Constraints in the First Stage Formulation

The objective function and the auxiliary constraints we propose to incorporate into the formulation of the first stage (i.e., (SG1)) are described in the following two subsections. Then, the resulted, complete first stage formulation is sketched at the end.

Note that both the objective function and the auxiliary constraints presented below are derived based on the initial partitioning of the linear system $Ax \leq b$, i.e., prior to the very first iteration of the proposed two-stage solution process, and will remain unchanged and simply be applied to all iterations afterwards. In theory, it would be ideal to update them accordingly as additional coverage constraints will be brought back to the first stage (i.e., based on the new partitioning) as the solution process proceeds. Such dynamic updates should be straightforward given our proposed design below, and we leave this extension for future exploration.

The Objective Function

We design an objective function $f(x)$ to incorporate into the first stage's model (SG1), so that we are fixing the assignments of the selected services L wisely, leaving sufficient room for the remaining services' assignments, and achieving more flexibility for satisfying the rest of coverage constraints that are restored during the second stage.

Note that it's not necessary to find an optimal solution to this $f(x)$ in the first stage as discussed previously. The purpose of having an objective function is to guide the solver to provide us with a high-quality solution, instead of an arbitrary one, if possible, so that we are less likely to encounter infeasibility during the second stage. In practice, besides an optimality gap bound, we also set a time limit for solving (SG1) during the first stage, to avoid spending a large amount of time on further improving the objective value.

The objective function $f(x)$ we propose consists of four parts:

(I) **Avoid assigning selected services to highly demanded cells**. Some (resident, time) combinations (i.e., the cells in the scheduling table in Figure 4.1) are required by a significant number of unenforced coverage constraints. If in the solution of the first stage the selected services are assigned to those cells, then it becomes very hard to satisfy the corresponding unenforced constraints during the second stage because those cells will be locked due to the partial fixing. Thus, we'd like to penalize making such assignments.

We focus on the lower bounds of the unenforced service coverage requirements (denote them as $\overline{Cov} \subseteq Cov$, which just corresponds to a subset of constraints $A^2x \leq b^2$ resulting from the initial partitioning by our proposed two-stage approach in Section 4.4.3). More specifically, we loop through each of the unenforced requirements $cv \in \overline{Cov}$, which explicitly is modeled in the following form as presented by (a4) in Appendix 4.7.1:

$$cv_{lb} \leqslant \sum_{r \in cv_R} \sum_{s \in cv_S} \sum_{t \in cv_T} X_{r,s,t} \leqslant cv_{ub}.$$

We evenly distribute its lower bound cv_{lb} to each involved assignments. That is, $cv_{lb}/(|cv_R| \cdot |cv_S| \cdot |cv_T|)$ is the contribution required from each involved assignment $X_{r,s,t}$, in order to ensure the lower bound of this requirement cv is satisfied.

Then for each cell (r,t), $\forall r \in R, \forall t \in T$, in the scheduling table, we calculate the accumulated contribution $O_{(r,t)}$ that is required from (r,t) across all of the unenforced coverage requirements:

$$O_{(r,t)} := \sum_{s \in S} \max_{\substack{cv \in \overline{Cov}: \\ r \in cv_R, s \in cv_S, t \in cv_T}} \frac{cv_{lb}}{|cv_R| \cdot |cv_S| \cdot |cv_T|}.$$

In other words, we first figure out the greatest contribution that is required for each specific assignment $X_{r,s,t}$ across all unenforced coverage constraints. Then, for each cell (r,t), its accumulated required contribution will be the summation over all services of the greatest contribution of the corresponding assignment.

We can see that the larger $O_{(r,t)}$ is, the more likely it is that cell (r,t) must be assigned to some other services (i.e., other than the ones in L) for satisfying the remaining unenforced requirements. Thus, we penalize by an amount $O_{(r,t)}$ in the objective function if we assign one of the selected service in L to resident r at block t.

(II) **Prefer assigning selected services in pairs**. More specifically, if we assign one of the selected services to a specific cell (r,t) in the scheduling table (Figure 4.1), we'd like to also assign a selected service to its *counterpart* cell $(r,\bar{t})$, so that the corresponding "full" month for resident r is taken by some selected service(s). Here, we group the set of blocks in the academic year in pairs (where each pair corresponds to a calendar month, if the blocks are half-months), while the counterpart block is thus defined as the other one in a specific pair (i.e., counterpart of $\bar{t}$ is either $t-1$ or $t+1$). In other words, for example, if a specific selected service in L is to be assigned to Resident-1 during the first half of July, then we prefer having a selected service also assigned to Resident-1 during the second half of July, to assigning it to Resident-1 instead during some other block, say, the first half of August. The reason for this assignment preference is that many services at UMMS are subject to a specific duration rule, which requires them to be always performed in a 2-block rotation, while such a rotation should never start from the middle of a month (i.e., from the second block in a specific pair). Therefore, assigning the selected services in L to paired cells will leave more room for completing the remaining assignments during the second stage, compared to locking two "halves" separately.

Similar to the previous bullet I, for each cell (r,t), we calculate the accumulated contribution, denoted as $\bar{O}_{(r,t)}$, across all duration-restricted services (denoted as $\bar{S} \subseteq S$) on its counterpart $(r,\bar{t})$:

$$\bar{O}_{(r,t)} = \sum_{s\in\bar{S}} \max_{\substack{cv\in\overline{Cov}: \\ r\in cv_R, s\in cv_S, \bar{t}\in cv_T}} \frac{cv_{lb}}{|cv_R|\cdot|cv_S|\cdot|cv_T|}.$$

We will then penalize an amount of $\bar{O}_{(r,t)}$ in the objective function if we assign one of the selected services to cell (r,t) but without making a similar assignment to its counterpart $(r,\bar{t})$.

(III) **Distribute the assignments of the selected services across all residents in a balanced way**. We have two hyper-parameters, B and O_{res}, that for each specific resident, if the number of selected services assigned to him/her during the whole academic year (in the first stage) exceeds B, then a penalty of O_{res} will be introduced into the objective function for each of the excessive assignments.

(IV) **Evenly distribute the assignments of the selected services across all time periods for each cohort (i.e., residents from the same program and at the same PGY level)**. Since in practice, the majority of the coverage requirements are specified to cohorts, and are applied to a specific single time period (i.e., $|cv_T| = 1$), we want to balance the assignment fixing across all time periods within each cohort as best as we can, in order to reduce the risk of tightening the space too much to meet the unenforced constraints.

Therefore, for each cohort ch, let l_{ch} and u_{ch} be variables respectively specifying the smallest and the greatest number of residents in the cohort who are assigned to the selected services, across all time periods. We penalize an amount of $O_{blk}\cdot(u_{ch}-l_{ch})$ in the objective function, where O_{blk} is a hyper-parameter from input.

Auxiliary Constraints

The auxiliary constraints are designed based on the service coverage requirements imposed on the non-selected services, i.e., constraints $A^2x \leq b^2$ that are skipped during the initial first stage, to further restrict the corresponding decision-making so that it is less likely to result in an invalid assignment fixing and an infeasible second stage.

The auxiliary constraints enforce those skipped service coverage requirements $\overline{Cov}$ in an aggregated manner, which thus work as a relaxation of system $A^2x \leq b^2$ (specified by the very first partitioning) but of a much smaller size. More specifically, for a specific group of residents $R_G \subseteq R$ (in practice, we specify R_G to be the residents in a group of cohorts, i.e., a specific combination of a set of programs and a set of PGY levels), we derive the following auxiliary constraint to further tighten the first-stage model:

$$lb_{R_G} \leq \sum_{r \in R_G} \sum_{s \in L} X_{r,s,t} \leq ub_{R_G} \qquad \forall t \in T.$$

Here,

$$lb_{R_G} := |R_G| - \sum_{s \in S \backslash L} \min_{\substack{cv \in \overline{Cov}:\ s \in cv_S, \\ R_G \subseteq cv_R,\ t \in cv_T}} \{ub_{cv}\}, \qquad ub_{R_G} := |R_G| - \sum_{s \in S \backslash L} \max_{\substack{cv \in \overline{Cov}:\ cv_S = \{s\}, \\ cv_R \subseteq R_G,\ cv_T = \{t\}}} \{lb_{cv}\}.$$

Basically, for each non-selected service $s \in S \backslash L$, the min term in lb_{R_G} (the max term in ub_{R_G}, respectively) above provides a theoretical upper bound (lower bound, respectively) on the number of residents in group R_G that can be assigned to service s at block t. More specifically, it loops through all of the unenforced coverage requirements in $\overline{Cov}$ whose corresponding upper bound (lower bound, respectively) is applicable to the assignments of residents R_G to service s at t, and finds the tightest one. Summing over these tightest upper bounds (lower bounds) then produces a valid upper bound (lower bound) on the number of residents in R_G that can be assigned to non-selected services at t. Consequently, the values lb_{R_G} and ub_{R_G} are, respectively, a valid lower bound and upper bound on the number of residents in R_G that can be assigned to some selected services at time period t, which just specify the proposed auxiliary constraint.

For instance, consider a group of 10 residents, and 4 non-selected services (i.e., $S \backslash L$): ER, PICU, NICU, and Cards. Suppose there are in total 4 unenforced coverage requirements comprising $\overline{Cov}$: service ER requires exactly 1 resident from this group at block t; exactly 2 residents from the group should be assigned to PICU at t or $t+1$ cumulatively; NICU requires at least 1 but no more than 2 residents at block t, while in total at most 3 residents from the group can be assigned to either NICU or Cards at t. Then, according to the above formulation, we have $lb_{R_G} = 10 - (1+2+2+3) = 2$, and $ub_{R_G} = 10 - (1+0+1+0) = 8$ for this resident group at block t, as it is clear that at least 2 but no more than 8 residents in the group can be assigned to selected services (i.e., other than these 4 services) at t.

Note that the auxiliary constraints derived based on the proposed method above are not guaranteed to be tight, because we only consider each single non-selected service individually when calculating the bounds lb_{R_G} and ub_{R_G}. For example, for the described instance above, a tighter lower bound $lb_{R_G} = 4$ is indeed valid. However, in practice, since most of the service coverage requirements are imposed on a single service, skipping the combinations of different non-selected services will not be a big deal for the bound calculation, and we are still able to derive sufficiently tight and effective constraints.

A sketch of the complete first stage formulation (SG1)

Additional Notation

H :	The set of resident cohorts (i.e., program, PGY level combinations).
$R(ch)$:	The set of residents in cohort $ch \in H$. $R(ch) \subseteq R$.

$Z_{r,t}$: Binary variables for $\forall r \in R, \forall t \in T$. 1 if one of the selected services is assigned to cell (r,t), while it is not the case to its counterpart $(r,\bar{t})$; 0 otherwise.

B_r : Non-negative continuous variables for $r \in R$. The excessive number (i.e., the amount over B) of assignments of selected services to resident r during the whole academic year.

Formulation

$$\min \sum_{r\in R, t\in T} O_{(r,t)} \cdot \left(\sum_{s\in L} X_{r,s,t}\right) + \sum_{r\in R, t\in T} \bar{O}_{(r,t)} \cdot Z_{r,t} + \sum_{r\in R} O_{res} \cdot B_r + \sum_{ch\in H} O_{blk} \cdot (u_{ch} - l_{ch})$$

s.t. $A^1 x \leq b^1$

(i.e., all constraints in Appendix 4.7.1 except the unenforced coverage ones) +

$$\sum_{s\in L} X_{r,s,t} - \sum_{s\in L} X_{r,s,\bar{t}} \leq Z_{r,t} \qquad \forall r \in R,\ \forall t \in T$$

$$\sum_{s\in L}\sum_{t\in T} X_{r,s,t} - B \leq B_r \qquad \forall r \in R$$

$$l_{ch} \leq \sum_{r\in R(ch)} \sum_{s\in L} X_{r,s,t} \leq u_{ch} \qquad \forall t \in T,\ \forall ch \in H$$

$$lb_{R_G} \leq \sum_{r\in R_G} \sum_{s\in L} X_{r,s,t} \leq ub_{R_G} \qquad \forall t \in T,\ \forall H' \subseteq H,\ R_G := \bigcup_{ch\in H'} R(ch).$$

4.7.3 An Theoretical Analysis on the Occurrence of Case 2 for the Infeasibility of the Second Stage

We first summarize some key properties of the formulation we proposed for solving our problem. Based on them, we show, by construction, that when the second stage (SG2) is infeasible, Case 2, i.e., its LP-relaxation actually being feasible, is possible. Then, we provide a brief analysis on why we rarely encounter this case in practice.

Properties of the Formulation

According to the detailed formulation provided in Appendix 4.7.1, the following two properties hold for our base model, assuming that we write it in the general form as in (IP) for simplicity:

- The feasible region is bounded by the unit hyper-cube, i.e., every feasible solution $x \in \mathbb{R}^n$ to (IP) or its relaxation satisfies $x \in [0,1]^n$, since the model consists of only binary variables.

- The coefficient matrix A is a $(0,\pm 1)$ matrix, i.e., every entry in A is either -1, 0 or 1, while the right hand side vector b is integral.

These two properties also hold for both the first stage formulation (SG1) and the second stage formulation (SG2). In particular, they are valid for the following intermediate formulation as well, where $x^* := (x_p^*, x_q^*)$ is denoted to be the solution to the first stage (SG1) as previous:

$$
\text{(Intm)} \qquad \begin{aligned}
\max_{y:=(y_p,y_q)} \quad & 0 \\
\text{s.t.} \quad & A_p^1 y_p + A_q^1 y_q \leq b^1 \\
& x_q^* - y_q \leq 0 \\
& y \text{ integer.}
\end{aligned}
$$

Note that although we are considering the scenario that the second stage (SG2) is infeasible, this intermediate formulation (Intm) is indeed feasible, since $y = x^*$ is clearly a valid solution.

Possibility in Theory

The following lemma and theorem show that when the second stage model (SG2) turns out to be infeasible due to an invalid assignment fixing, its LP-relaxation can still in theory be feasible. In other words, after incorporating the linear constraints $A_p^2 y_p + A_q^2 y_q \leq b^2$ (which satisfies the second property above), the polyhedron of the LP-relaxation of the above intermediate formulation (Intm) can be nonempty, while all of the integer points it contains will be pruned off.

Lemma 4.2. *Let P be the polyhedron of the LP-relaxation of the intermediate model* (Intm). *Suppose $x^* = (x_1^*, x_2^*, \ldots, x_n^*)^\top$ is the solution to the first stage model* (SG1). *Without loss of generality, we assume the first k coordinates of x^* are 1s, while the rest are 0s. That is, $x_1^* = x_2^* = \cdots = x_k^* = 1$, and $x_{k+1}^* = \cdots = x_n^* = 0$. Then, constraint $c := \{\sum_{i=1}^k y_i - \sum_{i=k+1}^n y_i \le k-1\}$ cuts off integer solution $y = x^*$ to* (Intm)*, but does not cut off any other vertex of P.*

Proof. Clearly, solution $y = x^*$ violates constraint c, and therefore, it is cut off. Next, we only need to show that no any other vertex of P is cut off by c.

We prove this by contradiction. Suppose there is another vertex of P, denoted as $\bar{y}$, violating constraint c as well. Then, let's consider the following polyhedron $\bar{P}$:

$$\bar{P} := \begin{cases} y_i \le 1 & \text{For } i = 1, 2, \ldots, k \\ y_i \ge 0 & \text{For } i = k+1, \ldots, n \\ \sum_{i=1}^k y_i - \sum_{i=k+1}^n y_i \ge k-1. \end{cases}$$

Clearly, $\bar{y} \in \bar{P}$. In addition, $\bar{P}$ is a polytope (i.e., bounded), as $0 \le y_i \le 1$ for $i = 1, 2, \ldots, n$ can be deduced from the constraints.

Let's consider a series of integer points $\{V^j\}$ for $j = 1, 2, \ldots, n$ defined as follows:

$$V^j := \begin{cases} V_i^j = x_i^* & \text{If } i \ne j \\ V_i^j = 1 & \text{If } i = j,\ j > k \\ V_i^j = 0 & \text{Otherwise.} \end{cases}$$

In other words, $\{V^j\}$ are the n corner vertices adjacent to x^* in the unit hyper-cube $[0,1]^n$, where each exactly has one coordinate (i.e., j) having the opposite value comparing to x^*. It is easy to check that these n integer points $V^j \in \bar{P}$. Furthermore, they are vertices of the polytope $\bar{P}$, because constraints $\{y_i \leq 1 \text{ for } i = 1, 2, \ldots, k; i \neq j\}$, $\{y_i \geq 0 \text{ for } i = k+1, \ldots, n; i \neq j\}$ and $\sum_{i=1}^{k} y_i - \sum_{i=k+1}^{n} y_i \geq k-1$ are active at V^j, and they are independent. Moreover, in the same way, we can check that $y = x^*$ is also a vertex of $\bar{P}$. Since polytope $\bar{P}$ is defined by, in total, $n+1$ constraints, it can have at most $n+1$ vertices. Therefore, we know that $\{V^j\}$ and x^* are the entire set of vertices describing polytope $\bar{P}$.

Since $\bar{y} \in \bar{P}$ and $\bar{P}$ is bounded (so there is no extreme ray), we know that, by the resolution theorem, there exists a convex combination $\bar{y} = \lambda_1 V^1 + \lambda_2 V^2 + \cdots \lambda_n V^n + \lambda_{n+1} x^*$, where $\sum_{m=1}^{n+1} \lambda_m = 1$ and $\lambda_m \geq 0$. Furthermore, we have $\lambda_{n+1} > 0$ because otherwise, $\bar{y}$ will be a convex combination of $\{V^j\}$, which are all right on the hyper-plane $\sum_{i=1}^{k} x_i - \sum_{i=k+1}^{n} x_i = k-1$, and thus $\bar{y}$ will be on this hyper-plane as well, and cannot violate constraint c.

Since $\bar{y}$ is a vertex of P, we know that there exists a constraint $\alpha y \leq \beta$, among the ones that describe polyhedron P, which is active on $\bar{y}$ but not active on x^* (since $y = x^*$ is clearly also a vertex of P). Suppose $\alpha x^* < \beta$ without loss of generality. Then, we claim that there exists a V^j such that $\alpha V^j > \beta$. By contradiction, if $\alpha V^j \leq \beta$ for all j, then $\alpha \bar{y} = \lambda_1 \alpha V^1 + \lambda_2 \alpha V^2 + \cdots \lambda_n \alpha V^n + \lambda_{n+1} \alpha x^* < \beta$, since $\lambda_{n+1} > 0$, which contradicts the fact that constraint $\alpha y \leq \beta$ is active at $\bar{y}$.

Denote $\alpha := (\alpha_1, \alpha_2, \ldots, \alpha_n)$. Then, we know that $\alpha_j \neq 0$ (because otherwise $\alpha x^* = \alpha V^j$, which is impossible), and there exists $0 < \gamma < 1$ such that $y' = \gamma x^* + (1-\gamma) V^j$ is right on the hyperplane $\alpha y = \beta$. More specifically regarding y', since V^j is only different from x^* on coordinate j, we have:

$$y' := \begin{cases} y'_i = x^*_i & i \neq j \\ y'_j = \gamma & \text{If } j \leq k \\ y'_j = 1-\gamma & \text{If } j > k. \end{cases}$$

In other words, y' has integer values on coordinates $i \neq j$, while have a fractional value on coordinate j (because $0 < \gamma < 1$). Since α has coefficients 0 or ± 1 (due to the second property) while $\alpha_j \neq 0$, we know that $\alpha y'$ is fractional. However, since β is an integer, this contradicts the fact that $\alpha y' = \beta$. Therefore, we can conclude that there is no other vertex of polyhedron P violating constraint c. □

Theorem 4.3. *If the polyhedron P, i.e., the LP-relaxation of the intermediate model* (Intm)*, is nontrivial (i.e., it has at least one non-integral vertex), then there exists a set of constraints $A'y \leq b'$, where A' is also a* $(0, \pm 1)$ *matrix while b' is integral, such that polyhedron $P' := \{y \mid y \in P,\ A'y \leq b'\}$ is nonempty, but it does not contain any integer points.*

Proof. This theorem is obvious given the above lemma. Let $A'y \leq b'$ be the linear system just comprising the constraint c (in the above lemma) for each of the integral vertices of P. Clearly, by this construction, A' and b' satisfy the required property. Then, the updated polyhedron P' of course does not contain any integer points, since they are all cut off by their respective constraint c in the system $A'y \leq b'$. Lastly, since P has a non-integral vertex (denoted as $\hat{y}$), which satisfies constraint c as indicated by the above lemma, and thus satisfies $A'\hat{y} \leq b'$, therefore we know that $\hat{y} \in P'$. Thus, P' is not empty. □

Analysis in Reality

The previous theorem implies that if we are extremely "lucky," e.g., if the unenforced constraints $A^2 y \leq b^2$ we restore during the second stage constitute exactly, or very close to, the system $A'x \leq b'$ constructed above, then we will encounter Case 2, where the second stage formulation (SG2) is infeasible but its LP-relaxation is feasible.

However, in practice, we rarely fall into this scenario. This is primarily because the constraints we bring back to the formulation during the second stage are all corresponding to service coverage rules, which have further properties than the general two listed at the beginning of this subsection. More specifically, the coefficients of coverage constraints are either $(0,1)$ or $(0,-1)$. However, this is not the case for the derived constraint

c in the lemma, as its coefficients are $(1,-1)$. On the contrary, those c's can often be viewed as the relaxation of our coverage constraints. More specifically, we can see that $\sum_{i=1}^{k} y_i \; - \; \sum_{i=k+1}^{n} y_i \; \leq \; k-1$ can be derived from $\sum_{i=1}^{k} y_i \leq k-1$, which is the relaxation of a potential coverage constraint $\sum_{i=1}^{k} y_i \leq ub$, since typically the upper bound value ub is much smaller then the "headcount." On the other hand, $\sum_{i=1}^{k} y_i \; - \; \sum_{i=k+1}^{n} y_i \; \leq \; k-1$ can be derived from $\sum_{i=k+1}^{n} y_i \geq 1$, which may further be the relaxation of a nontrivial coverage constraint $\sum_{i=k+1}^{n} y_i \geq lb$.

Therefore, unlike the derived c's which cut off just the "corners" of the hyper-cube without pruning any other vertices of the polyhedron P, when we restore the unenforced constraints (i.e., unenforced coverage rules) during the second stage, each of them may perform a much bolder cutting. Thus, besides the corresponding corner, a big chunk of P in the specified direction could be pruned off by each restored constraint, including the great amount of "symmetry" near that corner point and potentially also many other non-integral vertices of the polyhedron P. Thus, if at the end all integral vertices of P (the corners of the hyper-cube) are pruned off, then it's very likely that the whole updated polyhedron P' becomes empty (i.e., we will encounter Case 1, rather than Case 2).

4.7.4 Pseudo-code of a Power Iteration Method for solving the PageRank Model to Get the Service Pickiness Scores

Algorithm 5

```
1: Initialize $k = (\frac{1}{|V|}, \frac{1}{|V|}, \dots, \frac{1}{|V|})^\top$;
2: Initialize Boolean indicator $I = true$, and the stop threshold $\varepsilon = 10^{-4}$;
3: while $I$ do
4:     $k' \leftarrow \theta \bar{W} k + (1-\theta)\bar{h}$;
5:     if $|k - k'|_1 \leq \varepsilon$ then
6:         $I \leftarrow false$;
7:     $k \leftarrow k'$;
8: return $k$;
```

CHAPTER 5

Summary and Conclusions

In this dissertation, we studied in total three large-scale PSPs from two practical applications — two variations of cargo crew scheduling and medical resident block scheduling. We developed new modeling and/or solution approaches to more efficiently generate high-quality solutions to address the challenges of solving these problems.

In Chapter 2, we investigated crew pairing construction for a cargo airline. Due to the nature of its underlying flight network, the conventional subsequent crew rostering phase needs to be partially accomplished during this crew pairing construction process, which introduces a set of unique and complicated requirements. We proposed an SPPRC model to formulate the pricing problem of a delayed column generation framework to solve the crew pairing problem, which effectively incorporates all of those requirements into the identification of desirable crew pairings. We further developed several strategies and enhancements to accelerate the process of solving the proposed SPPRC model by a labeling algorithm. Experiments on real-world instances showed that our approach can solve the crew pairing problem in a much shorter time than a conventional DFS approach.

In Chapter 3, we studied an extension of the previous cargo crew pairing problem. A potential break in the middle of each crew pairing needs to be modeled in the construction process in order to improve the flight coverage achieved by the final crew pairing schedule. Although a straightforward modification of the proposed approach to the original crew pairing problem can be applied to solve this extension, solving the modified SPPRC model will take an extremely long time because a great number of additional arcs need to be introduced to update the network. To address this network density issue, we proposed an arc selection approach. By this approach, a significant number of arcs can temporarily be removed from the updated network for each specific iteration of the delayed column generation process, based on the results of a bidirectional labeling on the much sparser network model for the original crew pairing problem. Besides the significant improvement in the computational performance, we proved that this arc selection approach is an exact approach, which ensures the quality of the final output schedule. In experiments on practical instances from our partnered cargo airline, our approach was demonstrated to be able to solve this extended crew pairing problem in a short time while the achieved flight coverage was indeed improved by a great amount compared to the original problem without the break feature.

Lastly, in Chapter 4, we considered annual block scheduling for medical residents. Unlike traditional nurse shift scheduling, we must satisfy not only coverage requirements to guarantee an acceptable staffing level for different units (services) in the hospital, but also education requirements to ensure residents receive appropriate training to pursue their individual (sub-)specialty interests. This complex requirement structure, as well as the huge size and great amount of underlying symmetries, makes resident block scheduling a complicated combinatorial optimization problem. Solving a conventional integer program formulation for its practical instances directly using traditional MIP techniques will result in unacceptably slow performance. To address this computational challenge, we proposed a partial fixing approach, which completes the schedule construction iteratively through two sequential stages. The first stage focuses on the resident assignments for a small set of predetermined units (services) through solving a much smaller and easier problem relaxation, while the second stage completes the rest of the schedule construction after fixing those assignments specified by the first stage's solution. We developed cut generation mechanisms to prune off the bad decisions made by the first stage if infeasibility arises. We additionally proposed a network-based model to assist us with an effective unit (service) selection for the first stage to work on the corresponding resident assignments, in order to overall achieve an efficient and robust performance of the proposed two-stage iterative approach. We applied the proposed approach to solve real-world instances provided by our collaborating medical school. The experimental results demonstrated that our approach can solve the resident block scheduling problem in a dramatically shorter time than applying traditional MIP techniques directly to the integer program formulation.

Although the details of the proposed approaches and the associated logic in this dissertation are specified based on the practical operations of the cargo airline and the medical school we collaborated with, the general approaches and frameworks are flexible to incorporate additional features and requirements from the corresponding application background and can be adapted to solve problem variations and even other applications. More specifically, the flexibility of the proposed delayed column generation solution framework for cargo crew pairing is empowered by the SPPRC model. In generally, almost all common features and typical requirements in personnel scheduling and vehicle routing problems can be effectively incorporated through the underlying network design and its associated SPPRC modeling. Regarding medical resident scheduling, our proposed two-stage partial fixing approach is a general methodology, independent of any specific constraint structure once the problem can be formulated as a MIP base model. Therefore, this approach can be easily adapted to solve a wide range of PSPs from a variety of application areas, as the majority of rules and requirements in PSPs can easily be formulated in linear constraints. All of these demonstrate how the flexibility target we set with respect to the development of new solution models and approaches at the beginning of this dissertation (Chapter 1) has been achieved, in addition to the efficiency and solution quality achievements as highlighted in the respective chapters previously.

In future research, we plan to incorporate reserve crew scheduling into our cargo crew pairing construction phase so that a more realistic cost structure could be explicitly modeled to produce even better crew pairings for practical use. In addition, we'll consider incorporating other planning phases, e.g., fleet assignment and aircraft routing, into our crew pairing construction, to potentially further increase crew utilization and flight coverage and to improve the quality of the overall logistics schedule for the cargo airline. For resident block scheduling, we plan to design additional mechanisms to incorporate preferences from residents and the hospital administration explicitly into the proposed two-stage partial fixing construction process. We expect this way, a better-quality solution, rather than an arbitrary feasible solution, can be identified still within a reasonably short time, which can potentially reduce the needed iterations of "result, review, and revise" schedule construction process and overall speed up finalizing the schedule. Furthermore, we would like to apply our proposed approaches to other applications. For instance, we want to test whether our arc selection approach can also significantly reduce the solution time for long-haul vehicle/driver routing and how it impacts the corresponding solution quality. Additionally, we plan to apply our two-stage partial fixing framework to solve traditional nurse shift scheduling problems and trainer timetable construction problems and evaluate whether our approach is also beneficial to handle these much smaller, less complicated problems.

Moreover, to further increase the flexibility of our proposed approaches and address any remaining generalizability concerns, there are a couple of general feature research directions we can explore. First, we would like to consider how to expand the proposed arc selection approach for cargo crew pairing to work for more general setting where more than one break periods (or similar concepts in other applications) is allowed. A potential method we can explore is to recursively apply this proposed approach to one by one incorporate the potential breaks into the crew pairing construction. Second, for the proposed two-stage partial fixing approach, besides the incorporation of metrics and objectives explicitly into the iterative process as mentioned in the previous paragraph, we want to in addition consider generalizing it to be applicable to problems where the base model contains non-linear constraints. We would like to further develop and customize cut generation mechanisms depending on different types/features of the constraints that will not be enforced during the first stage to ensure that the overall efficiency is preserved for the corresponding non-linear scenario. This way, our two-stage iterative solution framework would be effective to more general problems, even for the ones from other applications outside the scope of personnel scheduling.

BIBLIOGRAPHY

Adamuthe, A. C. and Bichkar, R. (2011). Hybrid genetic algorithmic approaches for personnel timetabling and scheduling problems in healthcare. In *International Conference on Technology Systems and Management*.

Aggarwal, D., Saxena, D. K., Emmerich, M., and Paulose, S. (2018). On large-scale airline crew pairing generation. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 593–600. IEEE.

Aickelin, U., Burke, E. K., and Li, J. (2008). An evolutionary squeaky wheel optimization approach to personnel scheduling. *IEEE Transactions on evolutionary computation*, 13(2):433–443.

Aickelin, U. and Dowsland, K. A. (2004). An indirect genetic algorithm for a nurse-scheduling problem. *Computers & operations research*, 31(5):761–778.

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

Anbil, R., Tanga, R., and Johnson, E. L. (1992). A global approach to crew-pairing optimization. *IBM Systems Journal*, 31(1):71–78.

Arasu, A., Novak, J., Tomkins, A., and Tomlin, J. (2002). Pagerank computation and the structure of the web: Experiments and algorithms. In *Proceedings of the Eleventh International World Wide Web Conference, Poster Track*, pages 107–117.

Azaiez, M. N. and Al Sharif, S. (2005). A 0-1 goal programming model for nurse scheduling. *Computers & Operations Research*, 32(3):491–507.

Baker, E. and Fisher, M. (1981). Computational results for very large air crew scheduling problems. *Omega*, 9(6):613–618.

Balas, E. and Jeroslow, R. (1972). Canonical cuts on the unit hypercube. *SIAM Journal on Applied Mathematics*, 23(1):61–69.

Bard, J. F., Shu, Z., Morrice, D. J., and Leykum, L. K. (2017). Constructing block schedules for internal medicine residents. *IISE Transactions on Healthcare Systems Engineering*, 7(1):1–14.

Bard, J. F., Shu, Z., Morrice, D. J., Leykum, L. K., and Poursani, R. (2016). Annual block scheduling for family medicine residency programs with continuity clinic considerations. *IIE Transactions*, 48(9):797–811.

Barnhart, C., Hatay, L., and Johnson, E. L. (1995). Deadhead selection for the long-haul crew pairing problem. *Operations Research*, 43(3):491–499.

Barnhart, C., Johnson, E. L., Anbil, R., and Hatay, L. (1994). A column-generation technique for the long-haul crew-assignment problem. In *Optimization in industry 2*, pages 7–24. John Wiley & Sons, Inc.

Barnhart, C. and Shenoi, R. G. (1998). An approximate model and solution approach for the long-haul crew pairing problem. *Transportation Science*, 32(3):221–231.

Belien, J. and Demeulemeester, E. (2006). Scheduling trainees at a hospital department using a branch-and-price approach. *European journal of operational research*, 175(1):258–278.

Beliën, J. and Demeulemeester, E. (2007). On the trade-off between staff-decomposed and activity-decomposed column generation for a staff scheduling problem. *Annals of Operations Research*, 155(1):143–166.

Berrada, I., Ferland, J. A., and Michelon, P. (1996). A multi-objective approach to nurse scheduling with both hard and soft constraints. *Socio-Economic Planning Sciences*, 30(3):183–193.

Boeing (2018). World air cargo forecast 2018-2037, `https://www.boeing.com/resources/boeingdotcom/commercial/about-our-market/cargo-market-detail-wacf/download-report/assets/pdfs/2018_WACF.pdf`.

Borndörfer, R., Reuther, M., Schlechte, T., Waas, K., and Weider, S. (2016). Integrated optimization of rolling stock rotations for intercity railways. *Transportation Science*, 50(3):863–877.

Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine.

Brunner, J. O. and Edenharter, G. M. (2011). Long term staff scheduling of physicians with different experience levels in hospitals using column generation. *Health care management science*, 14(2):189–202.

Brusco, M. J. (1998). Solving personnel tour scheduling problems using the dual all-integer cutting plane. *IIE transactions*, 30(9):835–844.

Cacchiani, V. and Salazar-González, J.-J. (2017). Optimal solutions to a real-world integrated airline scheduling problem. *Transportation Science*, 51(1):250–268.

Chan, P., Heus, K., and Weil, G. (1998). Nurse scheduling with global constraints in chip: Gymnaste. In *Proc of Practical Application of Constraint Technology (PACT98), London, UK*.

Cheng, B. M. W., Lee, J. H. M., and Wu, J. C. K. (1997). A nurse rostering system using constraint programming and redundant modeling. *IEEE Transactions on Information Technology in Biomedicine*, 1(1):44–54.

Chinneck, J. W. and Dravnieks, E. W. (1991). Locating minimal infeasible constraint sets in linear programs. *ORSA Journal on Computing*, 3(2):157–168.

Cohn, A., Root, S., Kymissis, C., Esses, J., and Westmoreland, N. (2009). Scheduling medical residents at boston university school of medicine. *Interfaces*, 39(3):186–195.

Cohn, A. M. and Barnhart, C. (2003). Improving crew scheduling by incorporating key maintenance routing decisions. *Operations Research*, 51(3):387–396.

Conforti, M., Cornuéjols, G., and Zambelli, G. (2014). *Integer programming*, volume 271. Springer.

Deng, G.-F. and Lin, W.-T. (2011). Ant colony optimization-based algorithm for airline crew scheduling problem. *Expert Systems with Applications*, 38(5):5787–5793.

Derigs, U. and Friederichs, S. (2013). Air cargo scheduling: integrated models and solution procedures. *OR spectrum*, 35(2):325–362.

Derigs, U., Friederichs, S., and Schäfer, S. (2009). A new approach for air cargo network planning. *Transportation Science*, 43(3):370–380.

Desaulniers, G., Desrosiers, J., Dumas, Y., Marc, S., Rioux, B., Solomon, M. M., and Soumis, F. (1997). Crew pairing at air france. *European journal of operational research*, 97(2):245–259.

Desrochers, M. and Soumis, F. (1989). A column generation approach to the urban transit crew scheduling problem. *Transportation Science*, 23(1):1–13.

Desrosiers, J., Soumis, F., and Desrochers, M. (1984). Routing with time windows by column generation. *Networks*, 14(4):545–565.

Detienne, B., Péridy, L., Pinson, É., and Rivreau, D. (2009). Cut generation for an employee timetabling problem. *European Journal of Operational Research*, 197(3):1178–1184.

Dumas, Y., Desrosiers, J., and Soumis, F. (1991). The pickup and delivery problem with time windows. *European journal of operational research*, 54(1):7–22.

Dunbar, M., Froyland, G., and Wu, C.-L. (2012). Robust airline schedule planning: Minimizing propagated delay in an integrated routing and crewing framework. *Transportation Science*, 46(2):204–216.

Emden-Weinert, T. and Proksch, M. (1999). Best practice simulated annealing for the airline crew scheduling problem. *Journal of Heuristics*, 5(4):419–436.

Erdoğan, G., Haouari, M., Matoglu, M. Ö., and Özener, O. Ö. (2015). Solving a large-scale crew pairing problem. *Journal of the Operational Research Society*, 66(10):1742–1754.

Feillet, D., Dejax, P., Gendreau, M., and Gueguen, C. (2004). An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks: An International Journal*, 44(3):216–229.

Franz, L. S. and Miller, J. L. (1993). Scheduling medical residents to rotations: solving the large-scale multiperiod staff assignment problem. *Operations Research*, 41(2):269–279.

Freling, R., Lentink, R. M., and Wagelmans, A. P. (2004). A decision support system for crew planning in passenger transportation using a flexible branch-and-price algorithm. *Annals of Operations Research*, 127(1-4):203–222.

Gamache, M., Soumis, F., Marquis, G., and Desrosiers, J. (1999). A column generation approach for large-scale aircrew rostering problems. *Operations research*, 47(2):247–263.

Gamache, M., Soumis, F., Villeneuve, D., Desrosiers, J., and Gelinas, E. (1998). The preferential bidding system at air canada. *Transportation Science*, 32(3):246–255.

Gershkoff, I. (1989). Optimizing flight crew schedules. *Interfaces*, 19(4):29–43.

Gleeson, J. and Ryan, J. (1990). Identifying minimally infeasible subsystems of inequalities. *ORSA Journal on Computing*, 2(1):61–63.

Gopalakrishnan, B. and Johnson, E. L. (2005). Airline crew scheduling: state-of-the-art. *Annals of Operations Research*, 140(1):305–337.

Guieu, O. and Chinneck, J. W. (1999). Analyzing infeasible mixed-integer and integer linear programs. *INFORMS Journal on Computing*, 11(1):63–77.

Güler, M. G., İdi, K., Güler, E. Y., et al. (2013). A goal programming model for scheduling residents in an anesthesia and reanimation department. *Expert Systems with Applications*, 40(6):2117–2126.

Guo, J., Morrison, D. R., Jacobson, S. H., and Jokela, J. A. (2014). Complexity results for the basic residency scheduling problem. *Journal of Scheduling*, 17(3):211–223.

Guo, Y., Mellouli, T., Suhl, L., and Thiel, M. P. (2006). A partially integrated airline crew scheduling approach with time-dependent crew capacities and multiple home bases. *European Journal of Operational Research*, 171(3):1169–1181.

Guyon, O., Lemaire, P., Pinson, É., and Rivreau, D. (2010). Cut generation for an integrated employee timetabling and production scheduling problem. *European Journal of Operational Research*, 201(2):557–567.

Haouari, M., Zeghal Mansour, F., and Sherali, H. D. (2019). A new compact formulation for the daily crew pairing problem. *Transportation Science*, 53(3):811–828.

Irnich, S. and Desaulniers, G. (2005). Shortest path problems with resource constraints. In *Column generation*, pages 33–65. Springer.

Irnich, S., Desaulniers, G., Desrosiers, J., and Hadjar, A. (2010). Path-reduced costs for eliminating arcs in routing and scheduling. *INFORMS Journal on Computing*, 22(2):297–313.

Jaumard, B., Semet, F., and Vovor, T. (1998). A generalized linear programming model for nurse scheduling. *European journal of operational research*, 107(1):1–18.

Kahn, A. B. (1962). Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562.

Kasirzadeh, A., Saddoune, M., and Soumis, F. (2017). Airline crew scheduling: models, algorithms, and data sets. *EURO Journal on Transportation and Logistics*, 6(2):111–137.

Kim, K. and Mehrotra, S. (2015). A two-stage stochastic integer programming approach to integrated staffing and scheduling with application to nurse management. *Operations Research*, 63(6):1431–1451.

Klabjan, D., Johnson, E. L., Nemhauser, G. L., Gelman, E., and Ramaswamy, S. (2002). Airline crew scheduling with time windows and plane-count constraints. *Transportation science*, 36(3):337–348.

Koç, Ç., Jabali, O., and Laporte, G. (2017). Long-haul vehicle routing and scheduling with idling options. *Journal of the operational research society*, pages 1–13.

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

Leksakul, K. and Phetsawat, S. (2014). Nurse scheduling using genetic algorithm. *Mathematical Problems in Engineering*, 2014.

Li, D., Huang, H.-C., Chew, E.-P., and Morton, A. (2007). Simultaneous fleet assignment and cargo routing using benders decomposition. In *Container Terminals and Cargo Systems*, pages 315–331. Springer.

Liang, Z. and Chaovalitwongse, W. A. (2013). A network-based model for the integrated weekly aircraft maintenance routing and fleet assignment problem. *Transportation Science*, 47(4):493–507.

Maenhout, B. and Vanhoucke, M. (2010). Branching strategies in a branch-and-price approach for a multiple objective nurse scheduling problem. *Journal of Scheduling*, 13(1):77–93.

Miller, H. E., Pierskalla, W. P., and Rath, G. J. (1976). Nurse scheduling using mathematical programming. *Operations Research*, 24(5):857–870.

Ozdemir, H. T. and Mohan, C. K. (2001). Flight graph based genetic algorithm for crew scheduling in airlines. *Information Sciences*, 133(3-4):165–173.

Rahimian, E., Akartunalı, K., and Levine, J. (2017). A hybrid integer and constraint programming approach to solve nurse rostering problems. *Computers & Operations Research*, 82:83–94.

Rancourt, M.-E., Cordeau, J.-F., and Laporte, G. (2013). Long-haul vehicle routing and scheduling with working hour rules. *Transportation Science*, 47(1):81–107.

Rancourt, M.-E. and Paquette, J. (2014). Multicriteria optimization of a long-haul routing and scheduling problem. *Journal of Multi-Criteria Decision Analysis*, 21(5-6):239–255.

Rubin, J. (1973). A technique for the solution of massive set covering problems, with application to airline crew scheduling. *Transportation Science*, 7(1):34–48.

Ruther, S., Boland, N., Engineer, F. G., and Evans, I. (2016). Integrated aircraft routing, crew pairing, and tail assignment: branch-and-price with many pricing problems. *Transportation Science*, 51(1):177–195.

Şahin, G. and Yüceoğlu, B. (2011). Tactical crew planning in railways. *Transportation Research Part E: Logistics and Transportation Review*, 47(6):1221–1243.

Sandhu, R. and Klabjan, D. (2007). Integrated airline fleeting and crew-pairing decisions. *Operations Research*, 55(3):439–456.

Shao, S., Sherali, H. D., and Haouari, M. (2015). A novel model and decomposition approach for the integrated airline fleet assignment, aircraft routing, and crew pairing problem. *Transportation Science*, 51(1):233–249.

Sherali, H. D., Ramahi, M. H., and Saifee, Q. J. (2002). Hospital resident scheduling problem. *Production Planning & Control*, 13(2):220–233.

Syberfeldt, A., Andersson, M., Ng, A., and Bengtsson, V. (2015). Multi-objective evolutionary optimization of personnel scheduling. *International Journal of Artificial Intelligence & Applications*, 6(1):41–52.

Tamiz, M., Mardle, S. J., and Jones, D. F. (1996). Detecting iis in infeasible linear programmes using techniques from goal programming. *Computers & operations research*, 23(2):113–119.

Tang, C.-H., Yan, S., and Chen, Y.-H. (2008). An integrated model and solution algorithms for passenger, cargo, and combi flight scheduling. *Transportation Research Part E: Logistics and Transportation Review*, 44(6):1004–1024.

Topaloglu, S. and Ozkarahan, I. (2011). A constraint programming-based solution approach for medical resident scheduling problems. *Computers & Operations Research*, 38(1):246–255.

Trilling, L., Guinet, A., and Le Magny, D. (2006). Nurse scheduling using integer linear programming and constraint programming. *IFAC Proceedings Volumes*, 39(3):671–676.

Vance, P. H., Barnhart, C., Johnson, E. L., and Nemhauser, G. L. (1997). Airline crew scheduling: A new formulation and decomposition algorithm. *Operations Research*, 45(2):188–200.

Warburton, A. (1987). Approximation of pareto optima in multiple-objective, shortest-path problems. *Operations research*, 35(1):70–79.

Wei, K. and Vaze, V. (2018). Modeling crew itineraries and delays in the national air transportation system. *Transportation Science*, 52(5):1276–1296.

Weide, O., Ryan, D., and Ehrgott, M. (2010). An iterative approach to robust and integrated aircraft routing and crew scheduling. *Computers & Operations Research*, 37(5):833–844.

Weil, G., Heus, K., Francois, P., and Poujade, M. (1995). Constraint programming for nurse scheduling. *IEEE Engineering in medicine and biology magazine*, 14(4):417–422.

Wong, W. H., Zhang, A., Van Hui, Y., and Leung, L. C. (2009). Optimal baggage-limit policy: airline passenger and cargo allocation. *Transportation Science*, 43(3):355–369.

Yan, S., Chen, S.-C., and Chen, C.-H. (2006). Air cargo fleet routing and timetable setting with multiple on-time demands. *Transportation Research Part E: Logistics and Transportation Review*, 42(5):409–430.

Yan, S. and Tu, Y.-P. (2002). A network model for airline cabin crew scheduling. *European Journal of Operational Research*, 140(3):531–540.

Yao, Y., Zhao, W., Ergun, O., and Johnson, E. (2005). Crew pairing and aircraft routing for on-demand aviation with time window. *Available at SSRN 822265*.

Zhong, Q., Lusby, R. M., Larsen, J., Zhang, Y., and Peng, Q. (2019). Rolling stock scheduling with maintenance requirements at the chinese high-speed railway. *Transportation Research Part B: Methodological*, 126:24–44.