Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Master's thesis

Surgery planning using branch-and-price algorithm accelerated using machine learning

Bc. Pavlína Koutecká

Study programme: Open Informatics
Field of study: Bioinformatics
Supervisor: doc. Ing. Přemysl Šůcha, Ph.D.

Prague, May 2023

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Koutecká Pavlína** Personal ID number: **474383**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Bioinformatics**

II. Master's thesis details

Master's thesis title in English:

Surgery planning using branch-and-price algorithm accelerated using machine learning

Master's thesis title in Czech:

Plánování chirurgických zákroků pomocí metody branch and price zrychlené pomocí strojového učení

Guidelines:

The thesis aims to design a surgery planning algorithm. The purpose of the problem is to select patients requiring surgery and assign them to operating room blocks. Since it is a complex combinatorial problem, it is essential to design new methods able to reduce the computational time needed to solve the problem. An option is to use machine learning to improve solution-space exploration. The thesis comprises the following tasks:
1) Review the existing literature.
2) Analyse a real plan of surgeries and propose a generator of instances.
3) Design a branch and price algorithm for the studied problem.
4) Develop a machine learning technique to speed up the algorithm.
5) Evaluate the algorithm on generated instances and compare it with the results in the literature.

Bibliography / sources:

[1] Roman Václavík, Antonín Novák, Premysl Sucha, Zdenek Hanzálek:
Accelerating the Branch-and-Price Algorithm Using Machine Learning. Eur. J. Oper. Res. 271(3): 1055-1069 (2018)
[2] Mehdi A. Kamran, Behrooz Karimi, Nico P. Dellaert:
Uncertainty in advance scheduling problem in operating room planning. Comput. Ind. Eng. 126: 252-268 (2018)
[3] Frédéric Quesnel, Alice Wu, Guy Desaulniers, François Soumis:
Deep-learning-based partial pricing in a branch-and-price algorithm for personalized crew rostering. Comput. Oper. Res. 138: 105554 (2022)

Name and workplace of master's thesis supervisor:

doc. Ing. Přemysl Šůcha, Ph.D. Department of Control Engineering FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **06.02.2023** Deadline for master's thesis submission: _____

Assignment valid until: **22.09.2024**

doc. Ing. Přemysl Šůcha, Ph.D. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

. _____
Date of assignment receipt

_____
Student's signature

Declaration

I hereby declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the Methodical instructions for observing the ethical principles in the preparation of university thesis.

In Prague, 26 May 2023 .....

Pavlína Koutecká

Abstract

The scheduling of operating rooms in healthcare institutions is an essential task impacting the quality of patient care and operating expenses. Planning rules utilised in hospitals often result in suboptimal schedules. Optimisation techniques offer a promising solution. However, solving the operating room scheduling problem is challenging due to the large number of variables and constraints. This work addresses the problem by utilising the branch-and-price algorithm. Results show that most of the computation time is consumed by solving pricing problems. Thus, a machine learning model is proposed to accelerate the algorithm by analysing the structure of the problem and guiding the order for solving pricing problems. The algorithm is evaluated on synthetic instances generated based on real-world hospital settings. Results demonstrate significant improvements reducing more than 40 % of the number of solved pricing problems and over 10 % of computation time compared to the baseline method. This approach improves the efficiency of the branch-and-price algorithm and can be applied to other optimisation problems with similar characteristics.

Keywords: Operating room planning, branch-and-price, column generation, machine learning.

Abstrakt

Plánování operačních sálů ve zdravotnických zařízeních je zásadní úkol ovlivňující kvalitu péče o pacienty a provozní náklady. Plánovací pravidla používaná v nemocnicích často vedou k suboptimálním rozvrhům. Optimalizační techniky nabízejí slibné východisko. Řešení problému plánování operačních sálů je však obtížné kvůli velkému počtu proměnných a omezení. Tato práce řeší tento problém využitím algoritmu branch-and-price. Výsledky ukazují, že většina výpočetního času je strávena řešením pricing problémů. Proto je navržen model strojového učení, který algoritmus urychluje analýzou struktury problému a určením pořadí řešení pricing problémů. Tento algoritmus je hodnocen na syntetických instancích generovaných na základě nastavení reálných nemocnic. Výsledky ukazují významné snížení počtu vyřešených pricing problémů přes 40 % a snížení výpočetního času více než 10 % ve srovnání s původní metodou. Tento přístup zlepšuje efektivitu algoritmu branch-and-price a může být aplikován i na jiné optimalizační problémy s podobnými vlastnostmi.

Klíčová slova: Plánování operačních místností, branch-and-price, column generation, strojové učení.

Acknowledgements

I would like to express my heartfelt gratitude to my supervisor, doc. Ing. Přemysl Šůcha, Ph.D., for his unwavering guidance and support throughout my journey in completing this thesis. His expertise, patience, and commitment have been invaluable in shaping my research and helping me navigate the challenges along the way. I am grateful for the countless conversations we shared related to research, life and the world at large. It broadened my understanding, sparked new ideas and had a profound impact on me on a personal level.

I would also like to extend my sincere appreciation to MgA. Jan Hůla, Ph.D., for the insightful discussions we had, which have greatly influenced the development of my research.

Furthermore, I would like to express my gratitude to all the amazing individuals in the Industrial Informatics Research Center. The supportive spirit within the group has created an extraordinary environment to work in. The countless discussions and shared experiences have greatly impacted my growth and served as a constant source of motivation.

Last but certainly not least, I am profoundly thankful to all my friends, family and Vojta for their unwavering support throughout the studies. Their encouragement, belief in my abilities, and understanding during the challenging times have been fundamental to my journey. Their love and patience have shaped me into the person I am today, and I am forever grateful for their presence in my life.

Contents

List of Acronyms

AP average precision. 40, 41

DCG discounted cumulative gain. 40–42

GBDT gradient boosting decision trees. 39, 44, 47, 54

ILP integer linear program. 5, 35, 36, 52–54

LP linear program. 6, 33, 49

LTR learning to rank. 38

MART multiple additive regression trees. 44

MH master heuristic. 32, 33, 36

MILP mixed-integer linear program. 5

MIP mixed-integer program. 8, 12

MIQP mixed-integer quadratic program. 12, 17, 49, 63

ML machine learning. 6–9, 49, 60–62

MP master problem. 20, 31

MSS master surgical schedule. 2, 3, 11, 13

NDCG negative discounted cumulative gain. 40, 42, 47, 55, 56

OR operating room. 1–7, 11, 13, 16, 23, 37, 44, 50, 52–54, 56, 57, 59, 62–64

RMP restricted master problem. 21–23, 27, 28, 30, 31, 33–36, 45–47, 59, 60, 64

SHAP shapley additive explanations. 57–59

Chapter 1

Introduction

With the increasing demand for medical services healthcare institutions face a wide range of challenges, for example, rising costs, workforce shortages, or the need to adopt new technologies and innovative solutions to provide better care. In recent decades, many of these institutions have begun to deal with the issue of how to improve the quality of patient care while reducing operating expenses. Hospital managers are constantly looking for effective ways to minimise the required expenses and meet the needs of both hospital employees and patients. In this regard, we can identify operating room (OR) as a major cost driver, as it accounts for more than 40 % of the hospital's total expenses and a similarly large proportion of its total revenues [1]. This is caused both by the significant initial costs of surgical facilities and the costs of staff and equipment required to provide and maintain surgical services.

In this context, the scheduling of ORs is an important problem to study since it plays a crucial role in improving the quality of care and reducing costs of healthcare provision. Wisely chosen scheduling procedures in OR planning could, among others, lead to better utilisation of resources and shorter waiting time of patients, a higher number of performed surgeries, and, therefore, significant savings in time and money. However, nowadays, the vast majority of surgical planning in hospitals is done without optimised planning tools. Most of the planning is ruled by simple strategies stemming from common sense: operating room time is arranged in blocks, and these blocks are then allocated to different specialities according to their requirements on time and equipment. It is common that specific time slots are allocated to specific specialities just because it has always been done that way, regardless of the efficiency of these schedules.

The reason for using rather simple rules, resulting in schedules that are far from optimal, is that many variables are involved in the OR scheduling problem: preferences of each surgeon on time slots and ORs, limited resources, various surgery durations, and many more. To handle all of them as efficiently as possible, one has to move from paper-and-pen planning following simple rules to more advanced techniques. The field of optimisation can play a crucial role in addressing the challenges of scheduling surgeries in a hospital setting. By leveraging mathematical models and algorithms, optimisation approaches can help to balance many variables involved in the scheduling process and find efficient solutions that satisfy the needs of patients, surgeons, and hospital managers.

Nevertheless, from the optimisation point of view, the problem of ORs scheduling is also not

an easy task to solve. A mathematical formulation of a real-world problem which involves hundreds of patients and several weeks of planning horizon leads to an enormous number of variables, not speaking about all the constraints the model should reflect. Solving such tasks with traditional optimisation techniques would lead to hours of computation with an uncertain result. Moreover, in a real-life setting, where sudden situations often occur, such as an urgently admitted patient or an illness of a doctor, it is necessary to design flexible systems that can offer optimal or near optimal solutions in a limited time. To make decisions that would otherwise be computationally intractable, state-of-the-art optimisation algorithms nowadays heavily rely on manually designed heuristics. Machine learning methods offer a promising way for making such decisions in a more automated and optimised manner. By leveraging the power of machine learning, optimisation algorithms can learn from data and improve the ability to find high-quality solutions to challenging problems such as OR scheduling.

1.1 Motivation

An *operating room*, also known as an *operating theatre*, is an essential part of any hospital as it plays a crucial role in the diagnosis, treatment, and care of patients. It is where surgical procedures are performed, ranging from routine surgeries to complex and challenging operations, all performed by teams of surgeons, anesthesiologists, nurses, and support staff. Effective scheduling of ORs helps to ensure that surgical procedures are performed in a timely and efficient manner. To understand the characteristics of problems that hospitals encounter during operating room planning, we describe here the scheduling process from the hospital's management perspective.

The complex problem of operating room scheduling requires careful planning and coordination between multiple departments. It is commonly represented as a three-level decision-making procedure, including strategic, tactical, and operational level [2].

- At the *strategic decision level*, the number of operating rooms and working hours, together with the OR capacity assigned to each surgical specialty, are specified. This phase determines long-term decisions on the distribution of ORs by predicting the total demand of operating time for each department based on the past periods. Decisions on this level are made within a long planning horizon of several months to one year or longer.
- At the *tactical decision level*, a cyclic timetable, also known as master surgical schedule (MSS), is built on the medium-term standpoint following the decisions made on the strategic level. It divides the decided open time of available ORs into different surgeons or surgery groups over the scheduling window, typically in a range of weeks. This process is standardly reviewed every few months. An example of how the MSS might look like is shown in Figure 1.1.
- At the *operational decision level*, the last phase of the scheduling procedure, short-term decision-making is proceeded based on the constructed MSS. Surgeries from the waiting list are scheduled to specific OR, day, and starting time. The process is often divided into two stages. The first stage, known as the *advance scheduling problem*, involves selecting candidate patients from a waiting list of patients and assigning them

a surgery date and an OR. The second stage, referred to as the *allocation scheduling problem*, involves the sequencing of patients in each operating room on each day [3].

Upper management is typically responsible for making strategic decisions that affect the long-term future of the hospital. This may include decisions of opening a new cancer center or aligning the hospital with a regional healthcare system. Such decisions are critical to the overall success and growth of the hospital [4]. On the other hand, operational decisions related to the day of surgery, such as staff allocation, urgent case prioritization, and scheduling of additional cases, are typically the focus of clinicians following the long terms goals set by the upper management of the hospital. These decisions are short-term in nature and are aimed at ensuring the smooth and efficient functioning of the operating room from day to day.

Each hospital must also decide on the strategy of booking patients. Typically, hospitals adopt one of three main planning strategies: open scheduling, block scheduling or modified block scheduling [2].

- In the *open scheduling strategy*, patients are not classified into specialty groups. They are instead scheduled based on the convenience of surgeons and patients' priority, usually following the first-come-first-served rule.
- In the *block scheduling strategy*, the available capacity of ORs is divided into pre-defined time intervals (usually of half-day length) called blocks. Each block is then assigned to a particular surgeon or a group of surgeons based on the MSS, and they can arrange their surgical cases in this block as they wish.
- The *modified block scheduling strategy* is a combination of the first two policies. It assigns some of the blocks to specific specialties while keeping others open and releases unused capacity of blocks for other specialties at a later time.

In practice, the block scheduling strategy and modified block scheduling strategy are widely applied in hospitals [5]. That is because the preference of surgeons is to centralize their

	OR1		OR2		OR3		OR4		OR5	
	morning	afternoon	morning	afternoon	morning	afternoon	morning	afternoon	morning	afternoon
Monday	GEN	GEN	GEN	GEN	ORT	ORT	HEA	HEA	NEU	NEU
Tuesday	URO	GEN	GEN	GEN	ORT		HEA	HEA	OTO	OTO
Wednesday	URO	URO	GEN	GEN	OTO	OTO			NEU	NEU
Thursday	GEN		ORT	ORT	ORT	ORT	HEA	HEA	OTO	OTO
Friday	GEN	GEN	ORT	ORT	NEU	NEU	HEA		URO	

general surgery
urology
orthopedics
otorhinolaryngology
neurosurgery
cardiac surgery

Figure 1.1: An example of master surgical schedule for a time horizon of one week.

cases rather than to scatter them throughout the day. This makes scheduling easier since each surgeon has a fixed working time. Moreover, it reduces the complexity of scheduling problems as the surgeries only need to be allocated to blocks instead of specific time slots in the day [6].

The emphasis of this thesis is put on the lowest decision level of operating room scheduling, operation decision level. We aim to solve both discussed stages, namely assigning patients to ORs (advanced scheduling problem) and sequencing them (allocating scheduling problem). For that purpose, a block scheduling policy is assumed so that a given number of operating room blocks is assigned in advance to each surgeon.

1.2 Related work

This section aims at providing a comprehensive literature review on how operational research is applied to surgical planning and scheduling while also considering the potential for applying machine learning. We survey the literature across multiple fields related to this topic. At first, we focus on the methods used in OR scheduling. Particular attention is paid to the most prominent models and solution approaches used to address the problems arising in surgery scheduling. At second, we conduct a comprehensive literature review on the topic of the branch-and-price algorithm, as we use it to solve the problem in the following chapters of this work. Lastly, our focus is directed toward existing studies utilising machine learning techniques to address combinatorial optimisation problems.

1.2.1 OR planning and scheduling

In the past years, the area of surgery planning has received growing attention in the optimisation community. Several comprehensive articles are providing an in-depth review of the current state of research and methods in the field of operating room planning and scheduling, such as [2], [4], or [6]. As [6] stated, the total number of papers dealing with this topic is large. However, since the problem investigated in this work is focused on operating room planning at the operational level, in the following paragraphs, we concentrate only on the literature at the operational level. We discuss the relevant literature from the perspective of problem characteristics, objectives, and appropriate solution methodologies.

Problem characteristics

As mentioned in Section 1.2, OR scheduling problem on the operational level is typically approached as a two-stage problem – advance scheduling problem and allocation scheduling problem. In literature, authors mostly handle it as a two-step hierarchical method, where the problem can be broken down into two distinct yet interconnected problems that are solved separately [5]. In recent years, a limited number of articles also tackle the more complex problem of integrating both stages together, such as [7].

Objectives

In literature, studies approach the problem of OR scheduling from various angles, and many objectives are developed to model the preferences of different participants of the process. These include, for example, the cost of the performed surgeries [8], cost of resources [9], overutilisation and underutilisation [10], preferences of surgeons [11], patient urgency, or the number of patient waiting days [12]. Therefore, many papers model the OR scheduling as multi-objective optimisation problem. For instance, [13] specified a multi-criteria objective function that aimed to minimise the peak use of recovery beds, the occurrence of recovery overtime, and the violation of preferences of patients and surgeons. Differently, [14] focused on maximising capacity utilisation and minimising the risk of overtime, which could lead to cancelled surgeries. On top of that, [12] focused also on the number of scheduled patients and their priority. [15] sought to simultaneously maximise revenues and the total number of scheduled patients.

Solution methodologies

Various optimisation techniques, such as linear programming, mixed-integer programming, or dynamic programming, were introduced in the literature proposing exact solution methodologies. For example, [16] proposed an integer linear programming model to schedule elective surgeries from the waiting list on a weekly horizon. [17] presented a modified block strategy for the daily scheduling of elective patients modeled by mixed integer programming and constraint programming. Some works also aimed at the decomposition of the original formulation by reformulating, such as [7].

Apart from exact methodologies, heuristics approaches are also mentioned in the literature. For example, [18] tackled the problem through two sequential steps: first, they utilised column generation to address a linear program problem, and then employed a diving heuristic to obtain a feasible integer solution.

In recent years, research interest is moving towards more robust optimisation approaches, where, for example, the uncertainty of surgical durations is also considered. [3] proposed advance scheduling problem where uncertainty is considered via stochastic surgery duration. Different stochastic models were formulated and solved using the sample average approximation method and Benders decomposition technique. Other robust methods also include heuristics and metaheuristics techniques. In [14], various heuristics and local search methods that use statistical information on surgery duration were proposed to maximise capacity utilisation and minimise the risk of overtime and thus canceled surgeries. In [19], a two-level metaheuristic was developed to assign surgical cases to operating room blocks.

1.2.2 Branch-and-price in surgery scheduling

One of the long-established and commonly used combinatorial optimisation methods, which was not mentioned in the previous section, is the branch-and-price algorithm. This method aims to solve integer linear program (ILP) and mixed-integer linear program (MILP) problems with many variables. It has been successfully applied to a wide range of problems

including routing, transportation, resource allocation, and scheduling. Nowadays, there exists a growing interest in applying this algorithm to solve surgery scheduling problems, which are characterised by their high complexity and the need to consider multiple objectives.

[18] was among the first applications of the branch-and-price algorithm to surgery scheduling. In the study, they analysed a problem where a set of surgical cases is assigned to several operating rooms with the objective to minimise total operating costs. They first formulated this problem as an integer problem and then reformulated it using Dantzig-Wolf decomposition as a set partitioning problem. Based on this formulation, a branch-and-price exact solution algorithm was designed where columns for the linear program (LP) relaxation were created using a column generation algorithm. This approach was found to outperform existing methods in terms of solution quality and computational time. In [5], they followed up with a two-stage approach for an open-scheduling strategy. The first stage aimed at maximising the room utilisation via column generation, the second stage used a genetic algorithm for minimising the costs of idle time and overtime. Another paper on this topic is [13]. In contrast to [18], they studied how to sequence surgical cases in OR, i.e., how to arrange a set of surgical cases in a given operating room. To solve it, they applied a branch-and-price framework based on dynamic programming. On top of that, they elaborated on various branching strategies and branching schemes and examined their impact on the solution quality.

Recent works are based on the branch-and-price algorithm and develop more advanced variants. For example, [7] presented a branch-and-price algorithm extended by cutting plane procedures, called the branch-and-price-and-cut algorithm, that addresses both assigning set of patients to different ORs on different days and sequencing these patients to maximise the total scheduled surgical time. They also considered the maximum daily working hours of surgeons, prevented overlapping surgeries performed by the same surgeon, allowed time for obligatory cleaning when switching from infectious to noninfectious cases, and respected the surgery deadlines. In a similar manner, [20] also presented a branch-and-price-and-cut algorithm for the same problem. This paper considered a different model of the sequence-dependent operating room cleaning times that arise due to surgeries with varying levels of infection. In addition, human resources constraints were introduced.

Above mentioned studies demonstrated the potential of the branch-and-price algorithm to effectively solve complex surgery scheduling problems. However, certain factors can limit its effectiveness, including the size of the resulting mixed-integer program and the quality of the lower bounds generated through the column generation iterations. Thus, there is still a need for further research to develop more effective variants of the algorithm and machine-learning approaches hold promise for improvement.

1.2.3 Application of machine learning in combinatorial optimisation

Over the past few years, machine learning (ML) algorithms have gained increased attention from researchers for their potential to solve combinatorial optimisation problems. A clear proof of this trend is a recent study [21], which provides a comprehensive overview of ML techniques for combinatorial optimisation. It exhaustively surveys and highlights most of the recent attempts, both from the perspective of machine learning and operation research com-

munities, at how machine learning can leverage solving combinatorial optimisation problems. Since there are different strategies how to enhance combinatorial optimisation methods using ML, we limit our overview to techniques and applications close to our problem, therefore reviewing mainly machine learning applications in operating room planning, branch-and-price, and related methods. To better understand the topic of how ML can be applied in combinatorial optimisation, readers are encouraged to refer to [21].

With regard to existing studies combining OR scheduling and ML, there are few works so far. The main research is led towards estimating surgery case durations [22], [23], [24], which is critical for optimising operating room utilisation. Some articles, as [25], also try to predict which surgical cases have a potential risk of cancellation.

In terms of branch-and-price, there are several directions addressing the usage of machine learning. One can put his attention on the branching procedure as the choice of right variables to branch on can significantly reduce the final size of the branching tree. Several papers have covered this topic seeking to imitate strong branching, a powerful but computationally heavy branching rule consistently resulting in the smallest branching trees. [26] proposed a novel approach applying a graph neural network to learn branch-and-bound variable selection policies. The developed method involves constructing a graph representation of the combinatorial optimisation problem, where nodes represent the objects to be optimised, and edges encode the relationships between these objects. Other papers considering this topic are [27] and [28].

Another area of branch-and-price, where optimisation could be applied, is focused on column generation. [29] discussed the use of neural networks to predict optimal dual variables for the cutting stock problem. This information is then re-used for a solution algorithm, a stabilised column generation, to speed up the computation time.

Pricing problems are often a bottleneck of the algorithm as they are solved repetitively and can be extremely time-consuming. [30] stemmed from the idea that a pricing problem is repetitive in nature and the same problem is solved from scratch with a difference only in the input dual prices. They proposed to use a regression model learned from previous executions of the pricing problem for predicting a tight upper bound on the optimal value of the subproblem at each column generation iteration so that the solution space of pricing problems is reduced in future iterations. Another work incorporating the same idea is [31]. Some works aim to reduce the complexity of the subproblem by selecting only promising variables to perform the optimisation on. For example, [32] proposed a partial pricing scheme for a personalised aircrew rostering problem in which the subproblem only includes the variables likely to be selected in an optimal or near-optimal solution. The task of selecting the most promising variables is performed by a deep neural network trained on historical data. In this manner, a reduced subproblem is solved instead of the full one, decreasing the problem's complexity and execution time. [33] aimed at problems where the largest portion of the computing time consumes the subproblem. The core idea lies in reducing the searched network size by keeping only the most promising edges, which have a high chance of being a part of promising columns. Similar idea is also proposed by [34], [35] and [36].

Some works also study the impact of selected subproblems on the master problem. [37] presented a model behaving as a column selector designed for problems where a large portion of the computing time is spent in solving the master problem. In each column generation

iteration, they framed the column selection as a column classification task. Whether to select the column or not is determined by an ML model performing a one-step lookahead to identify the column that offers the maximum improvement to the master problem in the next iteration, thus speeding up the solving time.

1.3 Contribution

This work aims to design a surgery planning algorithm that addresses the problem of assigning patients to operating room blocks. We first express this problem as a mixed-integer program (MIP) and then reformulate it in terms of a branch-and-price algorithm that allows us to handle a large number of variables. Finally, we enhance the proposed algorithm by machine learning techniques.

The first contribution lies in an original formulation of the surgery scheduling problem considering multiple factors simultaneously, such as minimising patient waiting time, operating room idle and overtime, the number of days a surgeon must come to the hospital, and maximising the number of operated patients. Simultaneously to assigning patients to appropriate operating rooms, our goal is to optimise the sequencing of surgical procedures by taking into account sequence-dependent setup times. In other words, the stated formulation allows for solving both the advance scheduling problem and allocating scheduling problem at once.

As this is a complex combinatorial problem, it is essential to design new methods able to reduce the computational time spent on solving it. The main contribution of this work is then in reformulating the proposed problem in terms of the branch-and-price algorithm and the design of a novel application of machine learning, specifically aiming at enhancing the solution-space exploration process utilised in pricing problems. One of the common strategies for finding a new column for the master problem is to solve existing pricing problems until a column with a negative reduced cost is not found. However, we do not know in advance which of the pricing problems has the potential of generating a column with a negative reduced cost. To guide the search, we propose a machine learning-based "ranker" of the pricing problems. It analyses the structure of the instance being solved and, based on data collected from previously solved instances, ranks pricing problems according to the amount of negative reduced cost they are expected to obtain. Pricing problems are then solved in the order advised by the ranker.

Unlike some existing works that rely solely on end-to-end ML applications, our approach considers the underlying properties of the studied problem and combines this knowledge with ML. As mentioned in the previous subsection, the literature contains some papers that explore how to utilise the already gained knowledge from the solved pricing problems using machine learning. We do not try to reduce the number of column generation iterations, how it was proposed in [37], which describes the approach close to ours. Instead, we aim to reduce the number of pricing problems. Such approach is helpful for our and many other applications, where the majority of the computation time of the algorithm is spent solving pricing problems.

The last contribution lies in the development of a synthetic data generation method that reflects the real data with a high degree of accuracy. Traditionally, uniform distributions are

used in scheduling to generate parameters such as capacity, release time, and due date. By generating parameters from a uniform distribution, researchers can model a range of possible scenarios and test the robustness of their solutions under different conditions. However, there are numerous disadvantages to such an approach. A major one is that uniform distributions do not always accurately reflect real-world scenarios. Traditional scheduling algorithms do not suffer from this limitation; their complexity and process of finding solutions remain the same. However, for machine learning models, utilising parameters only from uniform distributions can be a huge drawback to their performance. By providing real-world data, ML can demonstrate its benefit over traditional scheduling methods as it may extract hidden patterns from the data and learn from them in a way that traditional scheduling algorithms never could. For that reason, we survey real data from hospitals to obtain credible parameters further used to estimate parameter distributions. With this information, we develop an automated process for generating instances that resemble real scenarios. That allows us to work with more instances, which can also be adjusted according to the specific requirements of the task at hand. This provides a unique contribution to the field of scheduling and machine learning.

Finally, the developed branch-and-price algorithm leveraged by a ML-based ranker is then tested on the generated synthetic instances. Results show that the proposed utilisation of machine learning significantly decreases the number of solved pricing problems and also the computing time spent on solving them compared to the baseline method without machine learning. In particular, we show that our approach is able to reduce the number of solved pricing problems by up to 48 % and the overall computation time by up to 17 %. As a result, our approach makes the branch-and-price more efficient and scalable. It can also be utilised in other applications, with the most notable results on problems where the computational time is spent mostly on solving the pricing problems.

1.4 Outline

The rest of the thesis is organized as follows. In Chapter 2, we introduce the addressed problem and present an integer program to solve it. Subsequently, Chapter 3 focuses on a general overview of the branch-and-price algorithm, also describing column generation and branch-and-bound methods, and applies it to transform the model introduced in Chapter 2. Additional improvements made on the branch-and-price algorithm are described here as well. In Chapter 4, we propose a machine learning framework for guiding the process of searching new variables in column generation. Chapter 5 provides a description of the data generation procedure and exhaustive experimental evaluation of the developed methodology. Finally, the thesis is concluded and ideas for future work are outlined in Chapter 6.

Chapter 2

Problem statement

In this work, we study the operating room scheduling problem consisting of two phases – assigning patients to operating rooms and putting them into a sequence accordingly. The general scheme of the assigning phase is as follows: select a set of patients from a waiting list of patients and assign them to a set of available operating room blocks in a given planning horizon, taking into account the availability of surgeons. We assume a block scheduling strategy, where the number and the length of available blocks are given. Since each block is related to a specific day, by assigning a patient to a block, the day of the surgery is fixed as well. In the sequencing phase, given the sets of patients assigned to operating room blocks, the aim is to determine the order in which the assigned patients will be operated within each operating room block. Such a sequencing is needed because the setup times to clean and prepare the room in between the surgeries are varying according to the type of performed surgery.

Several assumptions are made in regard to clarify the problem we study:

1. the type and duration of each surgery is estimated in advance and is deterministic,
2. emergency cases are not taken into account, i.e., the patient waiting list does not change during the scheduling horizon and the day each patient is ready to undergo surgery is determined in advance,
3. a single surgical specialty is considered,
4. surgeon for each surgical case is determined and cannot be changed,
5. a block scheduling strategy is used, i.e., a given number of operating room blocks is assigned to each surgeon in advance following the MSS,
6. the ORs are identical in terms of available time and equipment, i.e. a patient can be operated in any operating room in which the surgeon is available.

Note that, in usual practice, when a date is assigned to a surgical case, the operating theatre planners also specify the other members of the surgical team. The assumption is made that the human and instrumental resources, except for the surgeons, are always available whenever they are needed. This assumption can simplify the surgical planning process and avoid the need for extensive coordination. Nonetheless, it should be noted that the implicit allocation of team members can have some drawbacks such as reduced flexibility in adapting to unexpected events.

The high-level definition of the problem described above can be modelled as an integer programming problem. Specifically, in the remainder of this chapter, we introduce a mixed-integer quadratic program (MIQP) – a MIP model with a quadratic objective. We consider the model proposed in [3] as a baseline and introduce some modifications in order to make the model more realistic and applicable.

2.1 Notation

The following notations are used throughout the work.

Indices and sets:

$\mathcal{P}$	set of patients (index $p = 1, \ldots, P$),
$\mathcal{R}$	set of operating rooms (index $r = 1, \ldots, R$),
$\mathcal{B}$	set of blocks (index $b = 1, \ldots, B$),
$\mathcal{S}$	set of surgeons (index $s = 1, \ldots, S$),
$\mathcal{D}$	set of days in a planning horizon (index $d = 1, \ldots, D$),
$\mathcal{B}_d$	set of blocks in day d,
$\mathcal{B}_{rd}$	set of blocks in day d in room r,
$\mathcal{S}_b$	set of surgeons available in block b.

Parameters:

a_p	release time of patient p,
q_p	clinical priority coefficient of patient p,
t_p	expected surgery duration of patient p,
$t_{pp'}$	sequence dependent setup time between patient p and patient p',
l_{ps}	equals 1 if patient p needs to be treated by surgeon s; 0 otherwise,
d_b	day to which the block b is assigned,
c_b	capacity of block b,
$o_b^{\max}$	maximum overtime allowed by block b,
$o_r^{\max}$	maximum overtime allowed by operating room r,
v	maximum number of blocks assigned to a surgeon in any day,
v_s	maximum number of patients assigned to surgeon s in any day,
m_j	weight of term j ($j = 1, \ldots, 5$) in the objective function.

Variables:

$$x_{pb} = \begin{cases} 1 & \text{if patient } p \text{ is scheduled to be operated in block } b, \\ 0 & \text{otherwise,} \end{cases}$$

$$n_{sd} = \begin{cases} 1 & \text{if surgeon } s \text{ is scheduled for surgery in day } d, \\ 0 & \text{otherwise,} \end{cases}$$

$$y_{pp'b} = \begin{cases} 1 & \text{if patient } p \text{ is immediately followed by patient } p' \text{ in block } b, \\ 0 & \text{otherwise,} \end{cases}$$

$$e_b = \begin{cases} 1 & \text{if block } b \text{ is not empty,} \\ 0 & \text{otherwise,} \end{cases}$$

o_b amount of overtime of block b,

z_b amount of idle time of block b.

2.2 Mathematical formulation

Let us consider a set of planning days $\mathcal{D}$ and a set of parallel operating rooms $\mathcal{R}$ available on each day of the planning horizon. We consider a block scheduling strategy. Therefore, there is a set of blocks $\mathcal{B}$ splitting every room in each day into a morning and afternoon section so that there are $(2 \cdot D \cdot R)$ unique blocks in total. The regular capacity of a block is denoted by the parameter c_b. It is also assumed we have a set of surgeons $\mathcal{S}$. We suppose MSS as given. In other words, a set S_b of surgeons available to be assigned to block b is known a priori for each block in each day over the considered time horizon. Let $\mathcal{P}$ be a set of patients. The allocation of surgeons to the patients is known a priori and is given by parameter l_{ps}. All patients are described by a release time a_p, an expected surgery duration t_p, and a clinical priority coefficient q_p, which is used to distinguish the urgency of the surgery. The sequence-dependent setup time between every two patients is also determined in advance by a parameter $t_{pp'}$. The task is to select candidate patients from a set of patients $\mathcal{P}$, assign them to a proper operating room block from the set of blocks $\mathcal{B}$ and sequence the patients assigned to each block, taking into consideration all elements from the objective function described further in Section 2.2.1 and constraints described further in Section 2.2.2.

To formulate the task, we introduce a binary decision variable for assigning a patient to a specific block. We denote it by x_{pb}, and it yields 1 if the surgery of patient p can be performed in OR block b, otherwise, it is 0. To model the sequencing of patients, a binary variable $y_{pp'b}$ is used. It yields 1 if the surgery of patient p is immediately followed by the surgery of patient p' in OR block b, otherwise, it is 0. As the surgeon-patient assignment is given, we could avoid a variable for a surgeon in the formulation of the problem. However, the usage of such a variable in the model allows us to introduce additional constraints and objectives regarding the surgeons and their preferences. Therefore, we introduce a binary variable n_{sd}. It yields 1 if surgeon s is scheduled for at least one surgery in day d. Also, additional variables for the overtime and idle time of the blocks – o, z, respectively – reflect

the aspect of under and over-utilisation of operating rooms to the problem.

2.2.1 Objective function

Many studies have approached the studied problem from different perspectives, focusing either on the interests of hospital management, patients, or surgeons. The objective function we introduce combines multiple terms simultaneously so that none of the perspectives is neglected. This approach benefits from the improved flexibility of the model, allowing it to be customised to suit the specific requirements of different healthcare facilities and enabling optimised surgery planning in diverse contexts.

We aim at minimising the number of unscheduled patients (those not admitted to surgery within the planning horizon), overtime and idle time in operating rooms, the number of days patients have to wait for their surgery, and the number of days surgeons have to come to the hospital. To do so, we introduce an objective function that is made up of five terms as follows:

$$\begin{aligned}\min \quad & m_1 \left(\sum_{p\in\mathcal{P}} \sum_{d\in\mathcal{D}} \sum_{b\in\mathcal{B}_d} (d - a_p) q_p x_{pb} \right) - m_2 \left(\sum_{p\in\mathcal{P}} \sum_{b\in\mathcal{B}} x_{pb} \right) + m_3 \left(\sum_{s\in\mathcal{S}} \sum_{d\in\mathcal{D}} n_{sd} \right) + \\ & + m_4 \left(\sum_{b\in\mathcal{B}} o_b^2 \right) + m_5 \left(\sum_{b\in\mathcal{B}} z_b^2 \right).\end{aligned} \tag{2.1}$$

The first term represents the penalty of waiting days for scheduled patients. The term is weighted by the patient urgency parameter q_p so that the tardiness of urgent patients imposes a bigger penalty on the objective function. The second term is associated with the penalty for the number of unscheduled patients. The third term introduces preferences of the surgeons by penalising the total number of days the surgeons are planned to come to the hospital to accomplish their surgeries. The fourth term minimises the total overtime of blocks during the planning horizon, and, finally, the fifth term minimises the total idle time of blocks. The motivation for introducing the last two terms are the expenses associated with operating the room. The cost of each additional minute in the operating room is much greater than the cost of a regular one. Similarly, underutilisation of the operating room leads to financial losses, as fixed expenses such as rent and maintenance continue to increase even when the operating room is not in use [6]. It is important to observe that the expressions representing the overtime and idle time of the blocks are squared to impose a higher penalty on longer periods of under/over-utilisation, because the square function amplifies the penalty as the length of the period increases. In such a way, one longer under/over-utilisation is more costly than multiple shorter ones.

One problem arising with using multiple objectives simultaneously is that these objectives may have different scales or units, making it difficult to compare them directly. A common approach to address this issue is to homogenise the objectives by scaling them with appropriate coefficients. In our case, to obtain a well-balanced multi-objective function, each term is scaled by a coefficient $m_j, j = 1, \ldots, 5$.

2.2.2 Constraints

The first set of constraints we introduce concerns patients. The constraints should ensure that if a patient's surgery is scheduled, then it has to be in a block where the patient's surgeon is able to perform the surgery, and, at the same time, the surgery has to be scheduled after the release date of the patient. Constraint (2.2) guarantee that each patient receives surgical services at most once. Constraint (2.3) makes x_{pb} to be 1 only if a patient is scheduled to a block assigned to a surgeon that should treat the patient. Constraint (2.4) makes x_{pb} to be 1 only if the patient is available at the time when the block is planned.

$$\sum_{b \in \mathcal{B}} x_{pb} \leq 1, \qquad \forall p \in \mathcal{P}, \tag{2.2}$$

$$x_{pb} = 0, \qquad \forall p \in \mathcal{P}, b \in \mathcal{B} : \sum_{s \in \mathcal{S}_b} l_{ps} = 0, \tag{2.3}$$

$$x_{pb} = 0, \qquad \forall p \in \mathcal{P}, b \in \mathcal{B} : a_p > d_b. \tag{2.4}$$

Next, we need to constrain surgeons to show up in the hospital when the surgeries of their patients are scheduled. Constraint (2.5) makes n_{sd} to be 1 when a surgeon has surgery/surgeries in any of the blocks of a day. Parameter v_s additionally brings the restriction over the maximum number of surgeries per day allowed to be accomplished by a surgeon.

$$\sum_{b \in \mathcal{B}_d} \sum_{p \in \mathcal{P} : l_{ps} = 1} x_{pb} \leq v_s \, n_{sd}, \qquad \forall s \in \mathcal{S}, d \in \mathcal{D}. \tag{2.5}$$

Additional constraints limiting the time utilisation of operating rooms and blocks are also introduced. Constraint (2.6) forces the overtime of the block to be not more than the maximum permitted overtime of a block. Constraint (2.7) forces the overtime in each operating room and day to be not more than the maximum permitted overtime.

$$o_b \leq o_b^{\max}, \qquad \forall b \in \mathcal{B}, \tag{2.6}$$

$$\sum_{b \in \mathcal{B}_{rd}} o_b \leq o_r^{\max}, \qquad \forall r \in \mathcal{R}, d \in \mathcal{D}. \tag{2.7}$$

Next, we want to extend the problem by sequencing patients in blocks. One of the most important factors there is the surgery setup time, which is the time interval between surgeries required to clean, disinfect, and prepare the operating room for subsequent surgery. Usually, the setup time varies based on the type of the upcoming surgery and the preceding performed surgery. The goal is to optimise the sequencing of surgeries to minimise the total setup time between surgeries, as it potentially allows for more surgeries to be scheduled in the same operating room. This can be achieved by consecutively scheduling surgeries with similar equipment and setup requirements or by scheduling surgeries with longer setup times at the end of the day to minimise the impact on subsequent surgeries. To address this, we introduce constraint (2.8).

$$\sum_{p \in \mathcal{P}} t_p x_{pb} + \sum_{p,p' \in \mathcal{P} : p \neq p'} t_{pp'} y_{pp'b} - c_b = o_b - z_b, \qquad \forall b \in \mathcal{B}. \tag{2.8}$$

This constraint ensures that the total time spent on surgeries and set-up times for patients assigned to each block does not exceed the maximum capacity of that block while also considering overtime of the block. The left-hand side of the equation consists of three terms. The first term is the total time spent on surgeries in a block, the second term is the overall setup time between all pairs of subsequent patients in the block, and the third term represents the capacity of the block. The right-hand side consists of two terms – allowed overtime and idle time of the block. The idle time variable behaves here as a slack variable because it transforms the inequality constraint into equality.

Lastly, we constrain the precedence relationships in each OR block. A set $\{0, P{+}1\}$ represents the dummy patients, where patient 0 stands for the beginning of the block schedule and $P{+}1$ stands for the end of the block schedule. These dummy patients are introduced to ensure that every patient has a predecessor and successor so that there is no isolated patient. Constraints (2.9) and (2.10) ensure that every patient assigned to a block has exactly one predecessor and one successor. Constraints (2.11) and (2.12) say that if the block is not empty, i.e., at least one patient is scheduled to the block, exactly one patient has to be sequenced first (right after the dummy patient 0) and one patient has to be sequenced last (right before the dummy patient $P+1$). Constraints (2.13) and (2.14) enforce e_b to be 1 only if at least one patient is scheduled to the block.

$$\sum_{p' \in (\mathcal{P} \setminus p) \cup (P+1)} y_{pp'b} = x_{pb}, \qquad \forall p \in \mathcal{P}, b \in \mathcal{B}, \tag{2.9}$$

$$\sum_{p \in (\mathcal{P} \setminus p') \cup 0} y_{pp'b} = x_{p'b}, \qquad \forall p' \in \mathcal{P}, b \in \mathcal{B}, \tag{2.10}$$

$$\sum_{p \in \mathcal{P}} y_{p(P+1)b} = e_b, \qquad \forall b \in \mathcal{B}, \tag{2.11}$$

$$\sum_{p' \in \mathcal{P}} y_{0p'b} = e_b, \qquad \forall b \in \mathcal{B}, \tag{2.12}$$

$$x_{pb} \leq e_b, \qquad \forall p \in \mathcal{P}, b \in \mathcal{B}, \tag{2.13}$$

$$e_b \leq \sum_{p \in \mathcal{P}} x_{pb}, \qquad \forall b \in \mathcal{B}. \tag{2.14}$$

2.2.3 Final model

In conclusion, a single MIQP model (2.15) is constructed by combining the previously developed objective, constraints, and additional domain specifications for variables.

$$\min \quad m_1 \left(\sum_{p\in\mathcal{P}} \sum_{d\in\mathcal{D}} \sum_{b\in\mathcal{B}_d} (d - a_p) q_p x_{pb} \right) - m_2 \left(\sum_{p\in\mathcal{P}} \sum_{b\in\mathcal{B}} x_{pb} \right) + m_3 \left(\sum_{s\in\mathcal{S}} \sum_{d\in\mathcal{D}} n_{sd} \right) + \\ + m_4 \left(\sum_{b\in\mathcal{B}} o_b^2 \right) + m_5 \left(\sum_{b\in\mathcal{B}} z_b^2 \right),$$

subject to

$$\begin{aligned}
\sum_{b\in\mathcal{B}} x_{pb} &\leq 1, && \forall p \in \mathcal{P}, \\
\sum_{p\in\mathcal{P}} t_p x_{pb} + \sum_{p,p'\in\mathcal{P}:p\neq p'} t_{pp'} y_{pp'b} - c_b &= o_b - z_b, && \forall b \in \mathcal{B}, \\
o_b &\leq o_b^{\max}, && \forall b \in \mathcal{B}, \\
\sum_{b\in\mathcal{B}_{rd}} o_b &\leq o_r^{\max}, && \forall r \in \mathcal{R}, d \in \mathcal{D}, \\
\sum_{b\in\mathcal{B}_d} \sum_{p\in\mathcal{P}:l_{ps}=1} x_{pb} &\leq v_s\, n_{sd}, && \forall s \in \mathcal{S}, d \in \mathcal{D}, \\
x_{pb} &= 0, && \forall p \in \mathcal{P}, b \in \mathcal{B} : \sum_{s\in S_b} l_{ps} = 0, \\
x_{pb} &= 0, && \forall p \in \mathcal{P}, b \in \mathcal{B} : a_p > d_b, \\
\sum_{p'\in(\mathcal{P}\setminus p)\cup(P+1)} y_{pp'b} &= x_{pb}, && \forall p \in \mathcal{P}, b \in \mathcal{B}, \\
\sum_{p\in(\mathcal{P}\setminus p')\cup 0} y_{pp'b} &= x_{p'b}, && \forall p' \in \mathcal{P}, b \in \mathcal{B}, \\
\sum_{p\in\mathcal{P}} y_{p(P+1)b} &= e_b, && \forall b \in \mathcal{B}, \\
\sum_{p'\in\mathcal{P}} y_{0p'b} &= e_b, && \forall b \in \mathcal{B}, \\
x_{pb} &\leq e_b, && \forall p \in \mathcal{P}, b \in \mathcal{B}, \\
e_b &\leq \sum_{p\in\mathcal{P}} x_{pb}, && \forall b \in \mathcal{B}, \\
x_{pb} &\in \{0,1\}, && \forall p \in \mathcal{P}, b \in \mathcal{B}, \\
n_{sd} &\in \{0,1\}, && \forall s \in \mathcal{S}, d \in \mathcal{D}, \\
e_b &\in \{0,1\}, && \forall b \in \mathcal{B}, \\
y_{pp'b} &\in \{0,1\}, && \forall p \in \mathcal{P}, p' \in \mathcal{P} \setminus p, b \in \mathcal{B}, \\
o_b &\geq 0, && \forall b \in \mathcal{B}, \\
z_b &\geq 0, && \forall b \in \mathcal{B}.
\end{aligned} \tag{2.15}$$

Chapter 3

Branch-and-price

To achieve success in solving integer programming problems with a huge number of variables, such as the one presented in Chapter 2, it is crucial to use formulations where the linear relaxations provide tight approximations so that it can be combined with branch-and-bound, a popular method for solving integer linear programs, to obtain an integral solution. However, model (2.15) cannot be used directly because its linear relaxation is very weak, except when relatively small instances are considered. To overcome this difficulty, a powerful alternative, known as *branch-and-price*, should be employed.

Branch-and-price is a hybrid optimisation approach for solving large-scale integer linear programs that combines branch-and-bound, a popular technique for solving integer linear programs, with column generation, a technique for generating columns (variables) on-the-fly to strengthen the LP relaxations [38]. In column generation, the original problem is decomposed into a master problem and a pricing problem. Instead of handling large number of variables at once, sets of columns are left out of the master problem as most of these columns are likely to have their associated variable equal to zero in the optimal solution anyway. Then, the pricing problem is solved to identify columns with a profitable reduced cost. If such columns are found, they are iteratively added to the master problem, which is then reoptimised. Branching occurs when no profitable columns are found, but the solution fails to satisfy the integrality conditions. Branch-and-price uses column generation at every node of the branch-and-bound tree [39].

In the rest of this chapter, we will first describe the branch-and-price algorithm in general and then apply it to develop a new procedure for solving the model (2.15). The developed procedure will then be accelerated using standard techniques from the literature. Note that we provide here just a brief overview of the method. A comprehensive treatment of the branch-and-price method is provided in [38].

3.1 General scheme

As indicated above, numerous large-scale integer linear programming models are inapplicable in practice. Fortunately, many of them often exhibit a decomposable structure suitable for reformulations that allow to obtain stronger bounds or reduce symmetries. Dantzig and Wolfe [40] were among the first to exploit this idea in the following manner: the original problem,

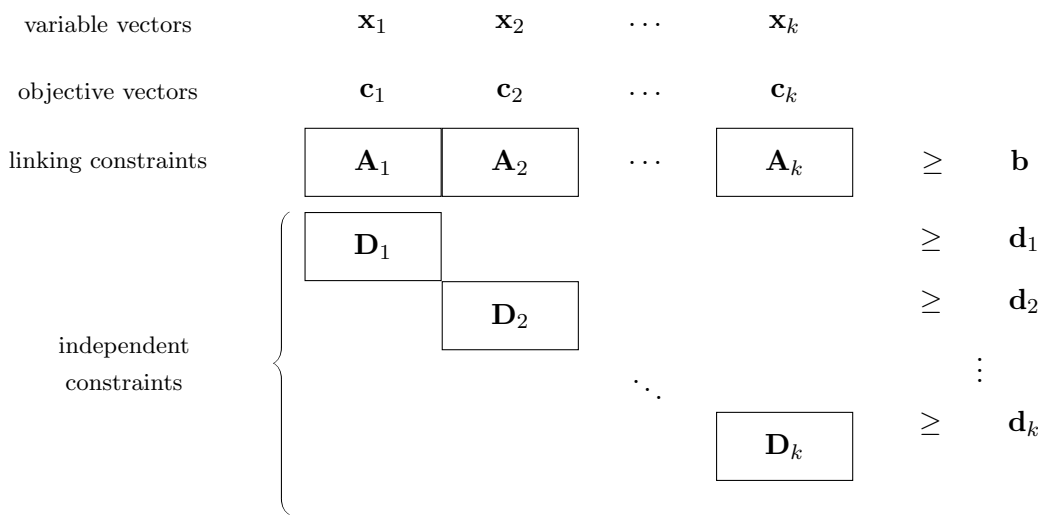


Figure 3.1: Schema of the Dantzig-Wolfe decomposition.

which may be too large and complex to solve, is broken down into smaller subproblems tied together by a relatively small set of constraints. These subproblems with original constraints are then solved individually on a smaller scale, leading to a more tractable solution [40].

Consider an integer program, called the *original problem*, of the form

$$\begin{aligned} \min \quad & \mathbf{c}^\top \mathbf{x}, \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b}, \\ & \mathbf{D}\mathbf{x} \geq \mathbf{d}, \\ & \mathbf{x} \in \mathbb{Z}^n_+, \end{aligned} \tag{3.1}$$

where $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{D} \in \mathbb{R}^{l \times n}$, $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{d} \in \mathbb{R}^l$, $m \in \mathbb{N}$, $n \in \mathbb{N}$, $l \in \mathbb{N}$. The decomposable structural property becomes apparent when the coefficient matrix of such a problem is arranged in a standard format, revealing a pattern similar to the one illustrated in Figure 3.1. In this figure, the constraint matrix is partitioned into non-zero blocks that consist of *linking constraints* $\mathbf{A}_1, \ldots, \mathbf{A}_k$, binding the columns together, and *independent constraints* $\mathbf{D}_1, \ldots, \mathbf{D}_k$, describing individual subproblems, together forming a block-diagonal matrix. Dantzig and Wolfe proposed that each of the k sets of independent constraints constitutes a subproblem of secondary importance to the whole program that should be studied separately [40]. The original formulation of the problem can be decomposed into the so-called *master problem*, taking control over the linking constraints in the matrix $\mathbf{A}$, and a *subproblem*, repeatedly solving individual problems respecting the constraint set $\mathbf{D}\mathbf{x} \geq \mathbf{d}$.

The linear program called *master problem (MP)* is defined as follows:

$$\begin{aligned} v(\text{MP}) := \min \quad & \sum_{j \in \mathcal{J}} c_j \theta_j, \\ \text{subject to} \quad & \sum_{j \in \mathcal{J}} \mathbf{a}_j \theta_j \geq \mathbf{b}, \\ & \theta_j \geq 0, \quad \forall j \in \mathcal{J}, \end{aligned} \tag{3.2}$$

where θ_j are variables associated with all feasible solutions within a solution space given implicitly by the original formulation and $\mathcal{J}$ is the index set of the variables θ_j. In many

applications, this often results in a huge master problem, as the number of variables grows exponentially with respect to the number of variables in the original formulation. In such a case, working with the master problem explicitly is not possible. However, given that only a few of these variables exhibit non-zero values in an optimal solution, it is reasonable not to consider all of them at once. Instead, we create a *restricted master problem (RMP)* by considering only a reasonably small subset $\mathcal{J}' \subseteq \mathcal{J}$ of these variables. From that, we iteratively insert new variables (columns) into the restricted master problem until reaching a point where one can prove that it represents an optimal solution for the master problem, requiring no further additional columns [38]. This is the basic principle of a *column generation* algorithm.

Let $\boldsymbol{\theta}$ and $\boldsymbol{\pi}$ be primal and dual solutions of the current RMP, respectively. The reduced cost $\bar{c}_j$ of variable θ_j is defined as

$$\bar{c}_j = c_j - \boldsymbol{\pi}^\top \mathbf{a}_j. \tag{3.3}$$

In a minimisation problem, a column is considered potentially beneficial for improving the current solution of the RMP if its reduced cost is negative. To find such a column, one solves the *pricing problem*, also called *subproblem*:

$$v(\text{PP}) \;:=\; \min_{j \in \mathcal{J}'} \{c_j - \boldsymbol{\pi}^\top \mathbf{a}_j\}. \tag{3.4}$$

When $v(\text{PP}) < 0$, the variable θ_j is considered potentially beneficial and its coefficient column $(c_j, \mathbf{a}_j)$ is added to the RMP. Note that, for some problems, the search for a column with negative reduced cost can be distributed across multiple pricing problems. Additionally, note that, as $\boldsymbol{\pi}$ also stands for dual prices of the constraints present in the master problem (3.2), the linking constraints of the original problem do not need to be explicitly present in the subproblem.

The process of solving alternately restricted master problem and subproblem is repeated until no further improving variable, i.e., no column with negative reduced cost, is found. At that point, an optimal solution to the RMP is obtained. If this solution is integral, it also represents a solution of the original formulation. If not, additional branching must be employed to obtain integer solutions. This approach of combining column generation with branch-and-bound is known as the *branch-and-price method* [41].

In the branch-and-price algorithm, the branching tree is constructed and searched using the branch-and-bound method. Each node of the tree represents a subtask of the initial problem created by branching on a variable. The search process in the branching tree starts at the root node, representing the initial restricted master problem. The column generation algorithm evaluates the linear relaxation of the problem. If the solution is integral, then this solution is feasible for the original problem, and the algorithm can terminate. If the solution is not integral, one has to select a fractional variable that is not yet fixed and branch on that variable by creating an additional integrality constraint. Accordingly, the search space is split into disjoint parts, and the column generation process is repeated recursively for each disjoint part until a node is reached where the relaxation has an integer solution.

The overall goal is to find the *incumbent solution*, an integer solution with the lowest objective

value among all the feasible integer solutions in the branching tree. To ensure the optimal integer solution is found, all the nodes in the branching tree yielding an integer solution must be explored. However, it can be a time-consuming procedure, as in the worst case, branching may require enumerating all possible solutions, which can be exponential in the number of variables. Nevertheless, in practice, many branches may be pruned by bounds obtained through column generation procedures:

1. *upper bound*: during the search process, the algorithm maintains an upper bound, which is the best integer feasible solution found so far. The upper bound is updated whenever a node is reached with a feasible integer solution with a lower objective value than the current upper bound. Often this bound is very tight and allows for efficient pruning.
2. *lower bound*: at each node, the solution of linear relaxation of RMP acts as a lower bound for the problem considering the active branching constraints. The current node can be fathomed whenever it exceeds the upper bound since a better integer solution has already been found elsewhere in the tree.

3.2 Baseline methodology

3.2.1 Column generation

To take advantage of the branch-and-price algorithm in our problem, we have to prepare the column generation by decomposing the original problem (2.15) to a master problem and subproblem via Dantzig-Wolfe decomposition. When decomposing, the first step should typically be to choose proper variables respecting the structure of the problem. In our scenario, the initial problem was allocating patients to blocks and determining their sequencing within each block. Attempting to solve this task for all blocks and patients simultaneously results in a highly complex problem with many variables. Nevertheless, this problem can be naturally divided into individual *blocks*. We propose to achieve the decomposition by transforming the original variables into a more compact and problem-suited set of variables on the block level. For that reason, we introduce a variable (column) that captures the relevant information about patients and their sequences for individual blocks. This variable is called a *pattern* and is defined as a possible sequence of patients assigned to a specific block satisfying maximum block overtime, surgeon availability and covering each patient at most once. Each pattern ω_j can be interpreted as a column of values for a specified block. The column includes the following indicators:

- patient indicators u_p, $p = 1, \ldots, P$: gets one if patient p is covered by pattern ω_j, otherwise, it is zero,
- surgeon indicators g_s, $s = 1, \ldots, S$: gets one if surgeon s is covered by pattern ω_j, otherwise, it is zero,
- block indicators w_b, $b = 1, \ldots, B$: gets one if block b is covered by pattern ω_j, otherwise, it is zero,
- overtime indicator h: indicates the amount of overtime in pattern ω_j,
- idle time indicator k: indicates the amount of idle time in pattern ω_j.

In this manner, a pattern can be represented as a single column by concatenating the values for the patient, surgeon and block indicators, and values of overtime and idle time. This vector would have a length of $(P + S + B + 2)$. An illustration of how some of the patterns might look like for a setup with three patients and two blocks is shown in Figure 3.2.

By introducing these new variables, patterns, we can decompose the original problem formulation into a master problem and subproblems as follows: the master problem will hold only the important connecting constraints, looking at the problem comprehensively to combine the patterns so that an overall solution for the entire system is obtained, whereas subproblems will take care of the actual scheduling and sequencing problems happening in each block independently. Specifically, in the OR scheduling, it means the master problem would hold the connecting constraints, ensuring that each patient is scheduled at most once, the maximal overtime in a room is respected, etc. In contrast, the subproblem would hold constraints for individual OR blocks, ensuring the overtime of a block is respected, a surgeon is not scheduled to the block if he is not available, patients are sequenced optimally, etc. This approach allows us to break down the initial complex task into smaller, more manageable subproblems, where the complexity of the solution space is significantly reduced.

Master problem

Let $\Omega = \Omega_1 \cup \ldots \cup \Omega_B$ be a union of feasible patterns for all the considered blocks in the time horizon. In the following, we call the master problem the Dantzig-Wolfe decomposition of the model (2.15) followed by its linear relaxation. As the size of set Ω grows exponentially with the number of patients, we introduce a $RMP(\overline{\Omega})$, the restriction of the master problem

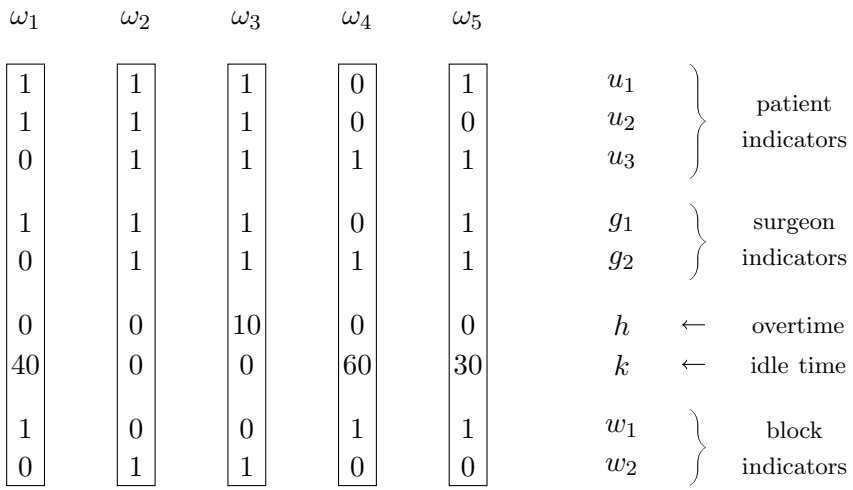


Figure 3.2: Example of five achievable columns $\omega_1, \ldots, \omega_5$ for an instance with three patients and two blocks. For example, pattern ω_1 assigns patient 1 and patient 2 to block 1 and sequences them so the idle time of block 1 is 40 units. The rest of the patterns can be described in a similar manner. It should be noted that although the indicators are the same for patterns ω_2 and ω_3, there is a difference in the amount of overtime. That is caused by a different sequencing of the patients.

to a subset of variables $\overline{\Omega} \subset \Omega$, in a following manner:

$$v(MP(\overline{\Omega})) := \min\ m_1 \left(\sum_{p\in\mathcal{P}} \sum_{d\in\mathcal{D}} \sum_{\omega\in\overline{\Omega}} (d - a_p) q_p u_{pd\omega} \theta_\omega \right) - m_5 \left(\sum_{p\in\mathcal{P}} \sum_{d\in\mathcal{D}} \sum_{\omega\in\overline{\Omega}} u_{pd\omega} \theta_\omega \right) + \\ + m_6 \left(\sum_{s\in\mathcal{S}} \sum_{d\in\mathcal{D}} n_{sd} \right) + m_7 \left(\sum_{r\in\mathcal{R}} \sum_{d\in\mathcal{D}} \sum_{\omega\in\overline{\Omega}} h^2_{rd\omega} \theta_\omega \right) + m_8 \left(\sum_{r\in\mathcal{R}} \sum_{d\in\mathcal{D}} \sum_{\omega\in\overline{\Omega}} k^2_{rd\omega} \theta_\omega \right), \tag{3.5}$$

subject to

$$\sum_{\omega\in\overline{\Omega}} w_{b\omega} \theta_\omega = 1, \qquad \forall b \in \mathcal{B}, \tag{3.6}$$

$$\sum_{d\in D} \sum_{\omega\in\overline{\Omega}} u_{pd\omega} \theta_\omega \leq 1, \qquad \forall p \in \mathcal{P}, \tag{3.7}$$

$$\sum_{\omega\in\overline{\Omega}} g_{sd\omega} \theta_\omega \leq v\ n_{sd}, \qquad \forall s \in \mathcal{S}, d \in \mathcal{D}, \tag{3.8}$$

$$\sum_{\omega\in\overline{\Omega}} h_{rd\omega} \theta_\omega \leq o^{\max}_r, \qquad \forall r \in \mathcal{R}, d \in \mathcal{D}, \tag{3.9}$$

$$\theta_\omega \geq 0, \qquad \forall \omega \in \overline{\Omega}, \tag{3.10}$$

$$n_{sd} \geq 0, \qquad \forall s \in \mathcal{S}, d \in \mathcal{D}, \tag{3.11}$$

where θ_ω indicates whether a pattern ω is selected ($\theta_\omega = 1$) or not ($\theta_\omega = 0$) in the solution; $u_{pd\omega}$ indicates if pattern ω contains patient p in day d ($u_{pd\omega} = 1$) or not ($u_{pd\omega} = 0$); $w_{b\omega}$ indicates whether pattern ω is in Ω_b ($w_{b\omega} = 1$) or not ($w_{b\omega} = 0$); $h_{rd\omega}$ indicates the amount of overtime of pattern ω if the pattern is related to room r in day d, otherwise it is zero ($h_{rd\omega} = 0$); $k_{rd\omega}$ indicates the amount of idle time of pattern ω if the pattern is related to room r in day d, otherwise it is zero ($h_{rd\omega} = 0$); $g_{sd\omega}$ indicates if pattern ω is related to surgeon s in day d ($g_{sd\omega} = 1$) or not ($g_{sd\omega} = 0$). Constraints (3.6) ensure that for every block exactly one pattern is used. Constraints (3.7) correspond to constraints (2.2) and guarantee that every patient is scheduled at most once. Constraints (3.8) correspond to constraints (2.5) and connect decision variable n_{sd} indicating if a surgeon is scheduled for surgery in a day with an analogical pattern's variable $g_{s\omega}$. Constraints (3.9) correspond to constraints (2.7) and connect maximum permitted overtime in each operating room and day with an analogical pattern's variable h_ω. Finally, constraints (3.10) and (3.11) indicate the domain of variables.

The question is to determine how a new variable should be generated in each iteration of column generation so that the restricted master problem solution is improved. The dual formulation of the master problem could help us here. Let $D(\overline{\Omega})$ be the dual program of $MP(\overline{\Omega})$:

$$v(D(\overline{\Omega})) := \max \sum_{p\in\mathcal{P}} \theta_p + \sum_{b\in\mathcal{B}} \mu_b + \sum_{r\in\mathcal{R}} \sum_{d\in\mathcal{D}} o^{\max}_r \xi_{rd}, \tag{3.12}$$

subject to

$$\sum_{p\in\mathcal{P}}\sum_{d\in\mathcal{D}} u_{pd\omega}\theta_p + \sum_{b\in\mathcal{B}} w_{b\omega}\mu_b + \sum_{r\in\mathcal{R}}\sum_{d\in\mathcal{D}} h_{rd\omega}\xi_{rd} + \sum_{s\in\mathcal{S}}\sum_{d\in\mathcal{D}} g_{sd\omega}\zeta_{sd} \leq$$
$$m_1\sum_{p\in\mathcal{P}}\sum_{d\in\mathcal{D}}(d-a_p)q_p u_{pd\omega} - m_5\sum_{p\in\mathcal{P}}\sum_{d\in\mathcal{D}} u_{pd\omega} + m_7\sum_{r\in R}\sum_{d\in\mathcal{D}} h_{rd\omega} + m_8\sum_{r\in\mathcal{R}}\sum_{d\in\mathcal{D}} k_{rd\omega} \quad (3.13)$$
$$\forall\omega\in\overline{\Omega},$$

$$-v\,\zeta_{sd} \leq m_6, \qquad \forall s\in\mathcal{S}, d\in\mathcal{D}, \quad (3.14)$$
$$\theta_p \leq 0, \qquad \forall p\in\mathcal{P}, \quad (3.15)$$
$$\xi_{rd} \leq 0, \qquad \forall r\in\mathcal{R}, d\in\mathcal{D}, \quad (3.16)$$
$$\zeta_{sd} \leq 0, \qquad \forall s\in\mathcal{S}, d\in\mathcal{D}. \quad (3.17)$$

In this model, θ_p is the non-positive dual variable associated with the surgical service of patient p (constraints (3.6)), μ_b is a dual variable associated with usage of block b (constraints (3.7)), ξ_{rd} is the non-positive dual variable associated with maximum overtime in room r in day d (constraints (3.8)) and ζ_{sd} is the non-positive dual variable associated with a maximum number of surgeries performed by surgeon s in day d (constraints (3.9)).

Moving back to the question of how new variables should be generated, the utilisation of the dual problem is as follows: we need to find a pattern so that a corresponding constraint (3.13) in the dual problem would be violated, making the dual problem infeasible. Such a constraint in dual formulation corresponds to the variable that could improve the solution of the master problem. Finding a variable like this requires the construction of a separate algorithm known as a pricing problem.

Summarising, the overall principle of the column generation method is to solve restricted master problem $RMP(\Omega)$, identify the violated dual constraints in $D(\overline{\Omega})$, i.e., primal variables (patterns) with a negative reduced cost, by solving pricing problems. Corresponding variables are then integrated into the set $\overline{\Omega}$. The process is repeated until the pricing problem solution proves that no violated constraint exists.

Pricing problem

The purpose of the pricing problem is to search for a pattern $\omega\in\Omega\backslash\overline{\Omega}$ so that for a given block b' in day d' and room r' holds

$$-\sum_{p\in\mathcal{P}}\theta_p\hat{x}_p - \mu_{b'} - \xi_{r'd'}\hat{o} - \sum_{s\in\mathcal{S}}\zeta_{sd'}\hat{n}_s +$$
$$+ m_1\sum_{p\in\mathcal{P}}(d'-a_p)q_p\hat{x}_p - m_5\sum_{p\in\mathcal{P}}\hat{x}_p + m_7\hat{o}^2 + m_8\hat{z}^2 < 0, \quad (3.18)$$

where $\hat{x}_p$ indicates whether patient p is selected or not, $\hat{n}_s$ determines whether surgeon s is assigned or not, $\hat{y}_{pp'}$ represents if the patient p is scheduled before patient p' in the block or not, $\hat{e}$ indicates if the block is selected or not, $\hat{o}$ represents the block's overtime and $\hat{z}$ represents the block's idle time. A pattern satisfying this condition is violating the dual constraint (3.13), which exactly corresponds to a primal variable with a negative reduced cost.

Following that, the pricing problem for a specific block b' in day d' and room r' can be formulated as:

$$\begin{aligned} v(PP) \;:=\; \min\; & -\sum_{p\in\mathcal{P}} \theta_p \hat{x}_p - \mu_{b'} - \xi_{r'd'}\hat{o} - \sum_{s\in\mathcal{S}_{b'}} \zeta_{sd'}\hat{n}_s + \\ & + m_1 \sum_{p\in\mathcal{P}} (d' - a_p) q_p \hat{x}_p - m_5 \sum_{p\in\mathcal{P}} \hat{x}_p + m_7\hat{o}^2 + m_8\hat{z}^2, \end{aligned} \tag{3.19}$$

subject to

$$\sum_{p\in\mathcal{P}} t_p\hat{x}_p - c_{b'} = \hat{o} - \hat{z}, \tag{3.20}$$

$$\hat{o} \le o_{b'}^{\max}, \tag{3.21}$$

$$\hat{x}_p \le \hat{n}_s, \qquad \forall p \in \mathcal{P}, s \in \mathcal{S}_{b'} : l_{ps} = 1, \tag{3.22}$$

$$\hat{x}_p = 0, \qquad \forall p : \sum_{s\in\mathcal{S}_{b'}} l_{ps} = 0, \tag{3.23}$$

$$\hat{x}_p = 0, \qquad \forall p : a_p > d', \tag{3.24}$$

$$\hat{n}_s = 0, \qquad \forall s \in \mathcal{S} \setminus \mathcal{S}_{b'}, \tag{3.25}$$

$$\sum_{p'\in(\mathcal{P}\setminus p)\cup(P+1)} \hat{y}_{pp'} = \hat{x}_p, \qquad \forall p \in \mathcal{P}, \tag{3.26}$$

$$\sum_{p\in(\mathcal{P}\setminus p')\cup 0} \hat{y}_{pp'} = \hat{x}_{p'}, \qquad \forall p' \in \mathcal{P}, \tag{3.27}$$

$$\sum_{p\in\mathcal{P}} \hat{y}_{p(P+1)} = \hat{e}, \tag{3.28}$$

$$\sum_{p'\in\mathcal{P}} \hat{y}_{0p'} = \hat{e}, \qquad \forall b \in \mathcal{B}, \tag{3.29}$$

$$\hat{x}_p \le \hat{e}, \qquad \forall p \in \mathcal{P}, \tag{3.30}$$

$$\hat{e} \le \sum_{p\in\mathcal{P}} \hat{x}_p, \tag{3.31}$$

$$\hat{x}_p \in \{0,1\}, \qquad \forall p \in \mathcal{P}, \tag{3.32}$$

$$\hat{n}_s \in \{0,1\}, \qquad \forall s \in \mathcal{S}_{b'}, \tag{3.33}$$

$$\hat{e} \in \{0,1\}, \tag{3.34}$$

$$\hat{y}_{pp'} \in \{0,1\}, \qquad \forall p \in \mathcal{P}, p' \in \mathcal{P} \setminus p, \tag{3.35}$$

$$\hat{o} \ge 0, \tag{3.36}$$

$$\hat{z} \ge 0. \tag{3.37}$$

The constraints are exactly the same as in the original formulation (2.15), yet, the complexity of the problem has been fundamentally reduced. We are now limited to one specific block only, so variables $\hat{x}_p$, $\hat{n}_{sd}$, $\hat{e}$, $\hat{y}_{pp'}$, $\hat{o}$, $\hat{z}$ are transformed to variables x_{pb}, n_{sd}, e_b, $y_{pp'b}$, o_b, z_b of the master problem respectively, but linked to a block. Therefore, the decomposition resulted in a subproblem of generating feasible patient schedules for individual blocks. In our case, the subproblem is usually viewed as a somehow modified version of the knapsack problem – we are trying to fit as many patients (*objects*) as we can, considering their urgency (*cost*), into one block (*knapsack*) so that the time spent on surgeries (*weight*) is minimal.

Note that the pricing problem is solved separately for each of the considered operating room blocks. Therefore, there are B pricing problems to solve in total. If the objective value of a subproblem is negative, a new variable with a column corresponding to this solution can be added to the RMP. In practice, there are multiple ways how to approach the process of adding new columns. Luckily, the success of the column generation method does not depend on selecting only the column with the highest reduced cost. Although identifying the column with the most negative reduced cost increases the probability that it will improve the master problem solution, it does not guarantee it. In fact, any column with a negative reduced cost could suffice. Thus, in some iterations of the column generation process, it may be enough to obtain a solution with a negative reduced cost rather than searching for the one with the most negative reduced cost. Essentially, we only have to iterate over all the pricing problems when proof of the optimality is needed (i.e., to prove that there are no columns with negative reduced cost). That allows for developing a wide range of ways to approach the search for new columns.

One of the most common approaches is to add the first column with a negative reduced cost encountered. This approach reduces the computation time required for each iteration, as it is not needed to solve every pricing problem. However, it may increase the number of iterations, so its overall effect is uncertain. Another possible alternative is to add a column with the highest reduced cost, which is motivated by the fact that the highest reduced cost is likely to significantly impact the master problem. However, this may not always be true. Additionally, it is necessary to iterate across all pricing problems to utilise this approach, which can be time-consuming. Another alternative is to choose all columns with a negative reduced cost encountered during the process. The impact on computation time is difficult to estimate. While it doesn't affect the time needed to solve the pricing problem, it may increase the time required to solve the restricted master and potentially increase or decrease the number of iterations needed.

3.2.2 Branching

In the previous section, we developed a column generation algorithm. However, the solution of linear relaxation of the restricted master problem is rarely integer-valued. Therefore, it is combined with a branch-and-bound procedure to produce integer solutions. The branching step is used to select a fractional variable from the current solution obtained from column generation, create branches based on the possible values that the variable can take on, and then resolve the problem again in each branch separately with the selected variable fixed. This process is repeated until an integer solution is obtained. Progressively, all the nodes produced by branching are explored to find the overall optimal solution.

However, some aspects of branching are still not very well defined, such as which variables to use for branching or how to search the branching tree. In the following paragraphs, we talk about these aspects of branching in detail.

Branching strategy

First, one must decide the strategy for branching. The most natural branching strategy would be to pick one of the fractional variables θ_k of the RMP and perform 0-1 branching, i.e., derive two branches where θ_k is respectively set to 0 (i.e., k-th pattern is not selected in the solution) and 1 (i.e., k-th pattern is selected in the solution). This, however, could be a bad choice for several reasons: (i) the variables in the RMP are only loosely connected to the variables of the original problem, (ii) fixed patterns could change the pricing problems, (iii) unbalanced tree may be created – when the pattern is fixed to zero (i.e. is not selected), it hardly reduces the search space as only a single pattern is prohibited.

To prevent these issues, we propose to use a different option: branching on original variables. Research [42] has shown that branching on the original variables often yields better results than branching on the master variables. We develop 0-1 branching decisions on original variables both for the patient variable x_{pb} and surgeon variable n_{sd}.

Regarding the *patient-block assignments*, fixing variable x_{pb} to zero ($x_{pb} = 0$) forbids patient p to be assigned to block b and fixing variable x_{pb} to one ($x_{pb} = 1$) requires patient p to be assigned to block b. To fix a patient p not to be assigned to block b, all columns associated with block b that have a one in the row corresponding to the patient's indicator u_p are removed. To fix a patient p to be assigned to block b, all columns associated with block b that have a zero in the row corresponding to the patient's indicator u_p are also removed.

It can be seen that the suggested branching scheme is compatible with the pricing problem. It allows to modify the pricing problem during column generation in a way that prevents the generation of infeasible columns resulting from the branching constraints. The pricing problem involves the solution of a knapsack problem for each block. Forbidding the assignment of patient p to block b is accomplished by involving additional constraint $\hat{x}_p = 0$ in the pricing problem of block b. Requiring the assignment of patient p to block b is accomplished by involving additional constraint $\hat{x}_p = 1$ in the pricing problem of block b.

Regarding the *surgeon-day assignments*, fixing variable n_{sd} to zero ($n_{sd} = 0$) forbids surgeon s to be assigned to day d and fixing variable n_{sd} to one ($n_{sd} = 1$) requires surgeon s to be assigned to day d. To fix a surgeon s to not be assigned to day d, all columns associated with day d that have a one in the row corresponding to surgeon's indicator g_s are removed. To fix a surgeon s to be assigned to day d, no columns need to be removed. Fixing in the case of surgeon-day assignments also have some implications on the dual program of master problem. Specifically, following the duality theory, (i) in case of $n_{sd} = 1$, the dual problem objective function (3.12) is extended with an additional term:

$$v(D(\overline{\Omega})) \;:=\; \max \sum_{p\in\mathcal{P}} \theta_p + \sum_{b\in\mathcal{B}} \mu_b + \sum_{r\in\mathcal{R}} \sum_{d\in\mathcal{D}} o_r^{\max} \xi_{rd} + v\, \zeta_{sd}, \tag{3.38}$$

(ii) both in the case $n_{sd} = 0$ and $n_{sd} = 1$, constraint (3.14) for surgeon s in day d is removed.

It might be considered to incorporate the presented branching constraints directly in the subproblem. However, that is not possible as a surgeon does not directly determine the pricing problem and patterns used in it. Therefore, for some surgeon-day combinations, the condition where a surgeon is assigned to a day may conflict with constraint (3.25). We could

think of skipping those subproblems for which the solution is infeasible, but the optimal solution may be missed by ignoring the infeasible subproblems. To prevent such problems, we directly add branching constraints for surgeon-day combination to the master problem.

Variable selection

In addition to the above-mentioned branching strategies, if there are more fractional variables simultaneously, one has to decide which variable to branch on next. According to [42], the first possible strategy for variable selection is to choose the most fractional variable to resolve the least-decided assignments first. Since we consider that values for variables x_{pb} and n_{sd} are between 0 and 1, it would mean selecting the variable with a value closest to 0.5. Another strategy would be to select a variable with the value closest to 1 to resolve the almost-decided assignments first.

However, in the given formulation, neither strategy provides a clear advantage or preference. For that reason, we proceed with a random strategy: among all the fractional variables, one is chosen randomly. This approach ensures fairness and avoids bias towards any particular variable. It is worth noting that even though there is no preference for most or least decided variables, the algorithm prioritises fractional surgeon-day assignments over patient-block assignments. This preference is based on the assumption that fixing a surgeon has a broader impact on all the patients allocated to that surgeon, making it a more "covering" constraint.

Branching scheme

We have already outlined how the current set of feasible solutions can be divided into smaller subsets via branching. Still, we have to specify how the next node to be solved is selected. Two common strategies used in searching the space are depth-first search and breadth-first search, each with its own advantages and disadvantages.

Depth-first search strategy explores the search tree in a depth-first manner. It starts in the root node and explores as far as possible along each branch before backtracking. Depth-first search is often used when the goal is to quickly find a feasible solution (i.e., integer solution), as experience has shown that feasible solutions are more likely to be found deeper in the tree rather than closer to the root [43].

Breadth-first search strategy explores the search tree in a breadth-first manner, exploring all the nodes at the current level before moving on to the next level. This ensures that the optimal solution, if it exists, will be found at the shallowest possible level, making breadth-first search more suitable for problems where the optimal solution is likely to be located at a higher level of the tree. However, breadth-first search tends to be slower compared to depth-first search, as it explores more nodes at each level before moving on to the next level [43].

It is known that the branch-and-price algorithm can be quite weak in generating good integer solutions quickly since the relaxed master problem solution is rarely integer-valued [44]. As a result, obtaining a good feasible solution as early as possible is the primary goal, preventing the generation of large branch-and-bound trees. For that reason, we choose to use the depth-first search strategy.

The whole procedure of branch-and-price described above is illustrated in Figure 3.3.

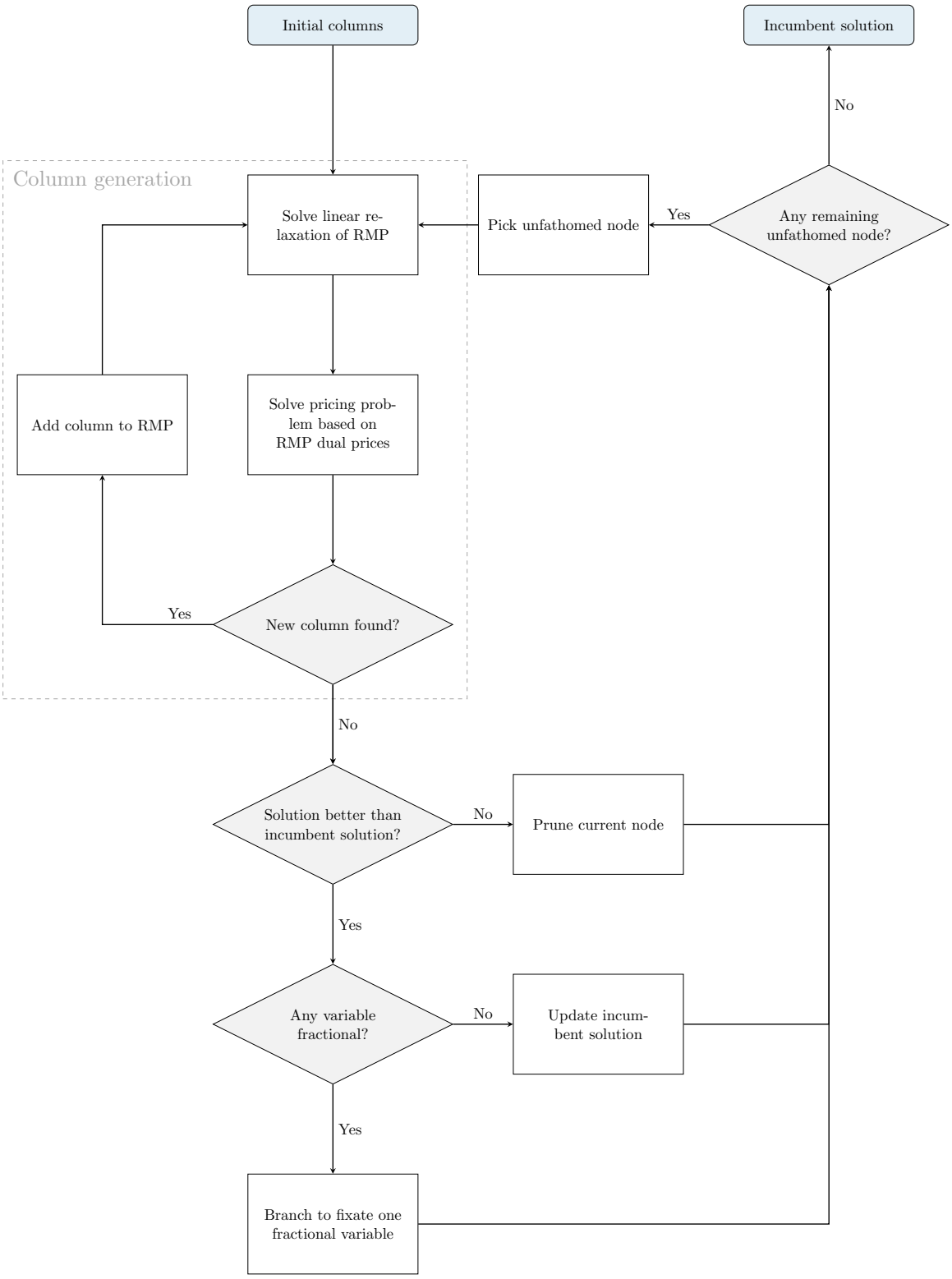


Figure 3.3: Schema of the branch-and-price algorithm.

3.3 Enhanced methodology

In many cases, acceleration techniques are crucial components that contribute to the effectiveness of the branch-and-price approach. Often these are simple but essential adjustments of the plain algorithm for obtaining high-quality solutions within a reduced time. In this context, we provide an overview of some of these strategies used in our method. We recommend referring to more comprehensive studies, such as [45], [46] or [39], for a general overview and in-depth understanding of presented techniques.

3.3.1 Initialisation procedure

Running the column generation, there are two critical situations leading to infeasibility we should be aware of: at the start, when no variables (columns) are generated, and after branching when removing some of the patterns according to the applied branching scheme [39]. Imagine a situation when there is an operating room block in the planning horizon in which, unfortunately, no surgeon is available to perform surgeries. Or maybe there is a block available to some surgeons but due to the branching constraints, all of them are forced not to operate in this block. Both of these situations lead to infeasible RMP even when the MP should have a feasible solution. The infeasibility is caused by constraint (3.6) in the RMP, which is forcing each block to be selected exactly once. Nonetheless, not every block necessarily has to be used in the optimal solution, some of them may stay empty. To address this issue, we generate a *trivial pattern* – a column with no assigned patients – for each block at the very beginning. This way, at least one feasible solution for the RMP is ensured comprising of all the trivial patterns.

One has to be careful with handling trivial patterns during the branching procedure, especially in the case of patient-block assignment. As described in Section 3.2.2, to fix a patient p to be assigned to block b, all columns associated with block b that have 0 in the row corresponding to patient p are removed. The rule must be followed by trivial patterns as well, but they cannot be eliminated easily. Thus, when a new branch where patient p is assigned to block b ($x_{pb} = 1$) is developed, the trivial pattern for block b is replaced with a so-called *minimal pattern*. In this pattern, the indicator for the assignment of patient p is changed from 0 to 1, the rest stays the same. The process is illustrated in Figure 3.4 on an instance with three patients, two blocks and two trivial patterns, where a new branching rule assigning patient 1 to block 1 is created.

Furthermore, in order to warm-up the column generation algorithm and generate an initial feasible solution that is not just trivial, a heuristic can be employed. The heuristic utilised in our case is a greedy one. Initially, patients are sorted based on their release dates. Then, for each patient, available blocks that the patient can be assigned to are retrieved and sorted based on their day. Subsequently, for each block, the capacity is checked to determine if it can accommodate the patient. If there is enough capacity, the patient is assigned to the block and sequenced after the last patient that was assigned to the same block. The heuristic iterates over all the patients. The whole process is described in Algorithm 1.

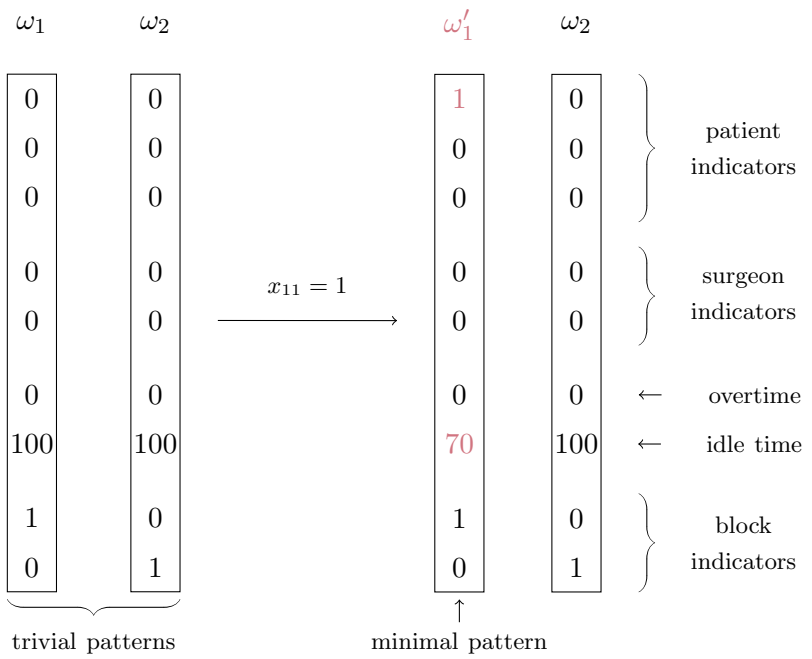


Figure 3.4: Process of converting a trivial pattern ω_1 to a minimal pattern ω_1' for an instance with three patients and two blocks when a patient-block assignment occurs in the branching procedure. In this example, an assignment of patient 1 to block 1 ($x_{11} = 1$) is applied. Therefore, indicator of patient 1 in ω_1' is fixed to 1, and, as both block here have a capacity of 100 units and the length of patient's 1 surgery is 30 units, the idle time of ω_1' is reduced to 70 units.

3.3.2 Master heuristic

The master heuristic (MH) is a technique that helps with obtaining a smaller branching tree. That is particularly useful in our use case, as we are working with quite weak branching conditions, and the branching tree might, eventually, get very large. MH can be utilised as follows: at the end of the column generation procedure, we set every θ variable in the master problem as a binary variable by editing constraint (3.10) to:

$$\theta_\omega \in \{0, 1\}, \qquad \forall \omega \in \overline{\Omega}. \tag{3.39}$$

We re-run the edited master problem forcing it to be solved as an integer program. Suppose the obtained solution is better than the incumbent solution (i.e., the objective value is lower). In that case, it is used to replace it since it is possible to find a better integral solution immediately. Otherwise, the obtained integer solution is neglected. The process of branching is then resumed. The procedure is described in Algorithm 2.

The advantage of this heuristic is that the provided solution may not be optimal, but it can often give a good upper bound for subsequent column generation runs. In addition, master heuristics can be used to find feasible solutions quickly in situations where finding the optimal solution is computationally intractable. The disadvantage is that it may increase the computational complexity of the problem. However, the modified problem can typically be solved in a reasonable time. Additionally, it can be accelerated using additional constraints on the objective value: the objective value has to be greater than the value obtained by

Algorithm 1: Initial feasible solution generation

```
Input: instance
Output: RMP
patterns ← ∅
patients ← sort instance.patients by release date
for patient in patients do
    blocks ← sort patient.blocks by day
    for block in blocks do
        if block.capacity ≥ patient.duration then
            add trivial pattern (patient, block) to patterns
            block.capacity ← block.capacity − patient.duration
            break
        end
    end
end
RMP ← load master problem together with patterns
```

Algorithm 2: Master heuristic

```
Input: RMP, incumbentSolution
Output: solutionMH
integerRMP ← RMP with integrality constraints on θ variables
solutionMH ← solveMasterProblem(integerRMP)
```

linear relaxation and lesser than the value of the known upper bound. This can lead to earlier termination as the search space is pruned.

3.3.3 Reduced cost fixing

Reduced cost fixing is other well-known technique used in LP-based branch-and-bound algorithms. Its main idea is based on reduced cost. Given a linear programming model and any optimal solution to this model, the reduced cost of a variable indicates the amount by which an objective function coefficient would have to improve before it would be possible for a corresponding variable to assume a positive value in the optimal solution. By the reduced cost of the variable, we can determine whether it has improvement potential on the objective value or not: if the reduced cost of the variable in the LP solution is greater than the difference between the LP's lower bound and upper bound, and the variable takes a value of 0, then it can be permanently fixed to 0 [45].

In our case, reduced cost fixing proceeds in each node of the branching tree straight after column generation for n_{sd} variables. Formally, for an RMP with current objective value z_{LP} as the lower bound, and incumbent integer solution objective value z_H as the upper bound, when for any n_{sd} variable with reduced cost $\overline{c}_{sd}$ following conditions hold:

$$z_{LP} + \overline{c}_{sd} > z_H, \tag{3.40}$$

Algorithm 3: Reduced cost fixing

Input: RMP, solutionRMP, incumbentSolution

Output: fixedRMP

fixedRMP ← *RMP*

for *surgeon in RMP.surgeons* **do**
  for *day in RMP.days* **do**
    reducedCost ← get reduced cost of (*surgeon*, *day*) variable from solutionRMP
    if *solutionRMP.obj + reducedCost > incumbentSolution.obj* **then**
      fix (*surgeon*, *day*) variable to 0 in *fixedRMP*
    end
  end
end

we can permanently set $n_{sd} = 0$ because fixing the variable to 1 would lead to a worse solution than the best integer solution so far. The procedure is described in Algorithm 3

The potential of this technique strongly depends on the quality of the provided RMP solution and the current incumbent solution. The closer the two bounds are, the more variables have the potential to be fixed.

3.3.4 Consequencing branching constraints

To speed up the branching process, we may take advantage of the branching rules developed for the patient-block or surgeon-day combination that does not satisfy the integrality condition.

In the case of the patient-block branching rule, for the branch where patient p is assigned to block b ($x_{pb} = 1$), an additional condition may be developed. If patient p is scheduled to a block, then also patient's surgeon $s = s(p)$ have to be scheduled to the same day:

$$n_{sd} = 1. \tag{3.41}$$

In the same manner, in the case of the surgeon-day branching rule, if surgeon s is not scheduled to a day d ($n_{sd} = 0$), then also none of his patients can be scheduled to any block in the same day:

$$x_{pb} = 0, \qquad \forall b \in \mathcal{B}_d, p \in \mathcal{P} : l_{ps} = 1. \tag{3.42}$$

These additional conditions can further help to reduce the size of the search space and speed up the branching process. The process is described in Algorithm 4.

3.3.5 Maximal branching depth

We have also developed a simple yet effective rule for the branching. Specifically, after entering every node, we check whether the depth of the current node in the branching tree exceeds a predefined threshold value. If it does, we halt the column generation procedure

Algorithm 4: Consequencing branching constraints

Input: RMP, constraint

Output: extendedRMP

extendedRMP ← *RMP* with *constraint* applied

if *(patient, block) variable is fixed to one in the constraint* **then**

    fix (*patient.surgeon*, *block.day*) variable to 1 in *extendedRMP*

end

if *(surgeon, day) variable is fixed to zero in the constraint* **then**

    for *patient in surgeon.patients* **do**

        for *block in day.blocks* **do**

            fix (*patient*, *block*) variable to 0 in *extendedRMP*

        end

    end

end

and, instead, we solve the original ILP model defined by (2.15) with additional constraints representing the branching decisions made up to the current node. The extension of the problem with additional branching decisions reduces the complexity of the initial problem that originally was hard to solve in a reasonable time. By solving the ILP model with these additional constraints, we immediately obtain an integral solution that is compared to the current incumbent solution and potentially updated if the new solution is better. Finally, we prune the node and move on to the next node in the branching tree.

This rule can significantly speed up the optimisation process by preventing the branch-and-price procedure from diving too deep into the branching tree. Furthermore, the threshold value can be adjusted to balance the tradeoff between the complexity of solved ILP and search efficiency. On the one hand, if the threshold is set too high, the algorithm may get stuck in a region deep in the branching tree where the optimal solution cannot be found. On the other hand, if the threshold is set too low, the number of developed branching constraints is small, resulting in an insignificant reduction in the complexity of the original problem and longer solving times.

3.3.6 Final overview

The resulting method with all the improvements is described in Algorithm 5. In particular, function **initialisation**() stands for the initialisation procedure described in Section 3.3.1, function **solveOriginalProblem**() represents the original problem indicated by model (2.15), function **solveMasterProblem**() represents the master problem as described in Section 3.2.1, function **solvePricingProblems**() represents the pricing problem as explained in Section 3.2.1, function **solveMasterHeuristic**() indicates the master heuristic procedure depicted in Section 3.3.2, function **reducedCostFixing**() stands for the reduced cost fixing procedure described in Section 3.3.3, and function **consequencingBranch**() represents the consequencing branching procedure described in Section 3.3.4.

Algorithm 5: Enhanced branch-and-price algorithm

Input: upperBound, maxDepth, instance

Output: incumbentSolution

RMP ← **initialisation**(*instance*) *incumbentSolution* ← *upperBound*
stack ← ∅
stack.push(*RMP*)
while *stack* ≠ ∅ **do**
    RMP ← *stack*.pop()
    if *currentDepth is higher than maxDepth* **then**
        solutionILP ← **solveOriginalProblem**(*RMP.branchingConstraints*)
        if *solutionILP.obj* < *incumbentSolution.obj* **then**
            incumbentSolution ← *solutionILP*
        end
        continue // `fathom current node in the branching tree`
    end
    do
        solutionRMP ← **solveMasterProblem**(*RMP*)
        column ← **solvePricingProblems**(*solutionRMP*)
        if *column.reducedCost* < *0* **then**
            add column to RMP
        end
    while *column.reducedCost* < *0*
    if *solutionRMP.obj* ≥ *incumbentSolution.obj* **then**
        continue // `fathom current node in the branching tree`
    end
    if *solutionRMP is integral* **then**
        incumbentSolution ← *solutionRMP*
    else
        solutionMH ← **solveMasterHeuristic**(*RMP*, *incumbentSolution*)
        if *solutionMH.obj* < *incumbentSolution.obj* **then**
            incumbentSolution ← *solutionMH*
        end
        fixedRMP ← **reducedCostFixing**(*RMP*, *solutionRMP*, *incumbentSolution*)
        constraint0, *constraint1* ← find a fractional combination and fix it to 0 and 1
        stack.push(**consequencingBranch**(*fixedRMP*, constraint0))
        stack.push(**consequencingBranch**(*fixedRMP*, constraint1))
    end
end

Chapter 4

Machine learning

In the previous chapter, we introduced the branch-and-price algorithm with several traditional techniques that can improve performance. However, the conventional methods for improving the branch-and-price algorithm have several drawbacks. Firstly, there is a lack of systematic techniques to improve algorithm performance on unseen instances by utilising the experience from past instances. This results in neglecting all the information acquired during previous runs when faced with a new instance. Secondly, developing efficient heuristic rules demands considerable time and effort for designing and testing. These observations could lead to the following question: is it possible to extract any useful information from solved instances and employ it effectively to accelerate the solving process on unseen instances?

One of the crucial components of the branch-and-price algorithm is the column generation procedure. The algorithm produces columns with negative reduced costs by iteratively solving the master problem and pricing problems. In our case, the pricing problem is a knapsack problem unique to each block, and even though it is a simplified version of the entire problem, solving it can still pose computational challenges. Moreover, from the nature of the problem, it is a repetitive task – similar pricing problems are solved repeatedly, often exhibiting minor variations. This raises the question of how to approach the process of searching for new columns smartly so that time is saved and information from previous runs is utilised.

In Section 3.2.1, we mentioned some traditional ways to approach the search for new columns. Most commonly, the space of pricing problems is either explored randomly until a column with a negative reduced cost is not found or is explored fully to obtain a column with the most negative reduced cost. However, in the second case, unfortunately, we must go through all the pricing problems, which takes a long time when there are many OR blocks. While we can shorten the traversal time by being satisfied with the first pattern with the negative cost found, we risk that such a column will not significantly impact the master problem. Furthermore, when, for example, there is only one column with a reduced cost, even random traversal can take a considerable amount of time.

To accelerate the procedure, we propose a new approach to search for improving patterns based on machine learning. This approach applies a trained model at each column generation iteration to predict the order in which the pricing problems should be solved. The order should reflect the reduced cost the column produced by the pricing problem will obtain. This way, we can effectively combine the benefits of multiple standard methods for searching the

space of pricing problems. The trained model ensures that columns with the most negative reduced costs (i.e., high-quality patterns) are added to the master problem while significantly reducing the search time, as the space of possible solutions is explored in an informed way based on the knowledge obtained from the previously solved instances. Moreover, the ranking can be performed in a very short computational time.

In fact, at each iteration of column generation, the fundamental question we ask is: *"Given the current solution of the master problem and all the possible pricing problems to be solved, what is the optimal order in which to solve them to obtain a pattern with the highest negative reduced cost as early as possible?"* Essentially, this is a ranking task where each pricing problem we encounter needs to be compared to other available pricing problems. In the following sections of this chapter, we give the necessary background: introduce the general concept of learning-to-rank problems, commonly used ranking algorithms for solving them, and quality functions to measure the performance. Then, we explain the process of gathering data to train the model. Finally, we describe the solution methodology employed to address the described task using the collected data.

4.1 Learning to rank

Learning to rank is an application of machine learning to solve ranking problems. It employs a supervised machine learning approach to learn from historic data and create a model that can generate a ranking for a given set of items according to their relevance to a given query [47]. In traditional supervised machine learning, a model is trained to predict a class or value for a single instance. For example, in a classification problem, the model is trained to classify a single instance into one of several predefined classes. Similarly, in a regression problem, the model is trained to predict a single value for a given instance. The main difference between mentioned approach and learning to rank (LTR) is that the latter deals with ranking a set of items and generating their *relative* order instead of predicting a single value for each instance *independently* of the others. That is more challenging than conventional supervised learning tasks primarily because of its hierarchical structure. For instance, a model that outputs scores of $-\infty, 0, +\infty$ for three items, respectively, would produce the same ranking as a model that outputs scores of $0.1, 2, 5$ for the same three items. LTR is particularly useful in search engines, where the goal is to provide users with the most relevant results for their queries. However, LTR can be helpful in any task where a ranked list of items should be produced [47].

The LTR problem is defined as follows: let q be a query sampled from a query distribution $\mathcal{Q}$. For this query, we are given a set of items $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ with each item $\mathbf{x}_i$ representing a vector of features, and a relevance vector $r = (r_1, \ldots, r_n) \in \mathbb{R}^n$. A learning-to-rank model is a function f that takes an item $\mathbf{x}_i$ and returns a score s_i. For query q, a vector of scores $\mathbf{s} = (f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)) \in \mathbb{R}^n$ is computed. To measure the performance of the model, i.e., how well the vector of scores $\mathbf{s}$ agrees with the relevance vector $\mathbf{r}$, we use a loss function $\mathcal{L}(f, q)$ such that $\mathbf{r}$ is contained in q, and $\mathbf{s}$ is computed internally. The goal of the LTR task is to, having Q queries $\{q_1, \ldots, q_Q\}$, find a model f from a given class of functions $\mathcal{F}$ so the

loss is minimised:

$$f = \arg\min_{f \in \mathcal{F}} \frac{1}{Q} \sum_{i=1}^{Q} \mathcal{L}(f, q_i). \tag{4.1}$$

Various machine learning methods, such as support vector machines, neural networks, or gradient boosting decision trees, can be used for the ranking task. Although neural networks have demonstrated significant achievements in various machine learning tasks, gradient boosting decision trees (GBDT) remain the leading algorithm for tabular datasets having diverse and noisy features, specifically outperforming other machine learning methods in the learning-to-rank problem [48]. That is mainly due to their ability to capture complex interactions between input features, handle missing values and handle both continuous and categorical variables. As a result, in the following, we will focus on a solution approach based on GBDT.

4.1.1 Gradient boosting decision trees

Gradient boosting decision trees is a popular supervised machine learning algorithm first introduced in [49]. It is an ensemble method combining multiple weak learners (in our case, decision trees) to form one strong learner. Like other boosting methods, the training of each weak learner depends on already trained learners. It works as follows: initially, a single learner (decision tree) is created. Subsequently, other weak learners (decision trees) are trained to correct the errors made by the previous learner and incorporated into it as "boosted" participants, together forming a strong learner. Specifically, each successive learner predicts the residuals (the difference between the actual and predicted values) of the preceding learner. The process continues until the ending conditions are not met [49].

Formally, let's assume a set of training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, where $\mathbf{x}$ is a vector of all the input variables and $\mathbf{y}$ is a vector of output variables. The goal is to train a model f to predict values of the form $\hat{\mathbf{y}} = f(\mathbf{x})$ by minimising an arbitrary differentiable loss function $\mathcal{L}(\mathbf{y}, f(\mathbf{x}))$. We start this process with a function model

$$f_0(\mathbf{x}) = \arg\min_{\gamma} \sum_{i=1}^{n} \mathcal{L}(y_i, \gamma), \tag{4.2}$$

where γ is a constant value. In each subsequent iteration j, the gradient boosting algorithm improves on the current model f_j by constructing a new model which adds an estimator h_j such that, together with f_{j-1}, it provides more accurate predictions in a form:

$$f_j(\mathbf{x}) = f_{j-1}(\mathbf{x}) + h_j(\mathbf{x}). \tag{4.3}$$

To achieve a more accurate prediction through $h_j(\mathbf{x})$, we should recall our ultimate objective: to utilise learner f in correctly predicting the output variable $\mathbf{y}$ so that $f(\mathbf{x}) = \mathbf{y}$ holds. Therefore, in our setting, the task of finding an optimal h_j during the j-th iteration reduces to fitting the learner to *residuals* in a form

$$r_{ij} = y_i - f_{j-1}(\mathbf{x}_i), \qquad i = 1, \ldots, n. \tag{4.4}$$

Unfortunately, finding an optimal h_j at each iteration for an arbitrary loss function might,

in general, be infeasible. Therefore, the idea is to, at least, minimise the difference between y and $f_{j-1}(x)$ as much as possible, such that

$$h_j = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{n} \mathcal{L}(y_i, f_{j-1}(\mathbf{x}_i) + h(\mathbf{x}_i)) \tag{4.5}$$

holds. To perform it, we find the local direction of the steepest descent on the loss function $\mathcal{L}(y, f_{j-1}(x))$, which is, in fact, a negative gradient of the function concerning the predictions of the current ensemble model, and move towards that direction. This is the core principle of the *gradient descent algorithm*. Using it, the estimator h_j is obtained by computing the gradients, also called *pseudo-residuals*:

$$p_{ij} = -\left[\frac{\partial \mathcal{L}(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}\right]_{f(\mathbf{x})=f_{j-1}(\mathbf{x})}, \qquad i = 1, \ldots, n, \tag{4.6}$$

and fitting $h_j \in \mathcal{H}$ to a new set of data $\{(\mathbf{x}_i, p_{ij})\}_{i=1}^{n}$. Note that this training set is similar to the original one, but the original labels y_i are replaced with the gradients p_{ij}.

Finally, the resulting estimator h_j is added to the current ensemble model f_{j-1} forming a better model f_j, and the process is repeated until a stopping criterion is reached. In practice, the stopping criterion can be based on the number of added trees, the improvement in the loss function, or the generalisation performance of the model on a validation set.

4.1.2 Ranking metrics

The choice of metric is critical for evaluating the effectiveness of ranking algorithms. It depends on the application and the problem at hand. In this section, we will focus on three widely used metrics in learning to rank: average precision (AP), discounted cumulative gain (DCG), and negative discounted cumulative gain (NDCG).

Average precision

AP is a widely used metric in information retrieval to evaluate the quality of a ranking algorithm working with *binary relevance,* i.e. when the relevance score r_i of an item x_i can be only 0 (item is not relevant) or 1 (item is relevant) [47]. It measures the average precision at different *ranks*. The precision at a particular rank k is defined as the fraction of relevant items among the top k items returned by the algorithm. The average precision is the average of the precision scores at all ranks where relevant items are retrieved.

Specifically, in order to obtain *precision* for a given query q, we measure out of top k retrieved items, how many of them are relevant:

$$\text{P@}k(q) = \frac{1}{k} \sum_{i=1}^{k} r_i. \tag{4.7}$$

By computing (4.7) for $k = 1, \ldots, n_q$, where n_q is the total number of items returned by the

algorithm for query q, we obtain the *average precision*:

$$\mathrm{AP}(q) = \frac{1}{\sum_{i=1}^{n_q} r_i} \sum_{k=1}^{n_q} \mathrm{P@}k(q) \cdot r_k. \tag{4.8}$$

A drawback of AP is that it does not consider the order of the retrieved items, only if the relevant items are present. In some cases, this may lead to inaccuracies and decreased performance. For example, consider an engine that retrieves ten items in response to a user query. If there are five relevant items in total, and the search engine retrieves all five of these, the AP score will be the same, regardless of the order in which the items appear. In this case, a ranking metric that takes into account the position of the relevant items may provide a more accurate evaluation.

Discounted cumulative gain

DCG is another widely used metric in learning-to-rank problems. It is used for tasks with binary relevance as well as with *graded relevance*, i.e. when the relevance score r_i of an item x_i is a discrete value in a defined range indicating the degree of relevance, for example, 0 (bad), 1 (fair), 2 (good), 3 (excellent), 4 (perfect) [47]. The essential principle of DCG is that more relevant items should be ranked above irrelevant ones. In that manner, it overcomes the drawbacks of AP explained above.

The *discounted cumulative gain* for a given query q is defined as

$$\mathrm{DCG}(q) = \sum_{k=1}^{n_q} \frac{2^{r_k} - 1}{\log_2(k+1)}, \tag{4.9}$$

where with the numerator, we measure how relevant the item is (the *gain*), and with the denominator, we measure a penalty for the rank of a retrieved item (the *discount*). Altogether, DCG value goes high when relevant items are ranked high (i.e., the higher the DCG, the better the ranking algorithm).

It is worth noting that using a variant of DCG, referred to as DCG@k, is also common in many applications. DCG@k calculates the DCG only up to a fixed rank k, corresponding to the maximum number of items a user is willing to examine. Therefore, only the top k relevant items contribute to the final calculation. It is often used in real-world applications, where users typically look at only a few top-ranked items. The formula for DCG@k is simply a truncated version of equation (4.9):

$$\mathrm{DCG@}k(q) = \sum_{i=1}^{k} \frac{2^{r_i} - 1}{\log_2(i+1)}. \tag{4.10}$$

One of the biggest drawbacks of the DCG metric is that, because of its cumulative nature, as the length of recommended items increases, there is a high chance that DCG increases as well. This means a longer recommendation list can have a higher DCG score than a shorter list, even if the shorter list contains better recommendations. To overcome this drawback, the concept of a normalised version is utilised.

Normalised discounted cumulative gain

NDCG is an extension of DCG that normalises the score by dividing it by the maximum achievable DCG score for a given set of recommendations [47]. In this manner, it considers that the maximum possible score should also depend on the number of relevant documents. Formally, NDCG is given as:

$$\text{NDCG} = \frac{\text{DCG}}{\text{IDCG}}, \tag{4.11}$$

where IDCG is the ideal DCG score defined as the maximum possible DCG value that can be obtained for a given set of items. It is computed by applying (4.9) for the ideal relevance scores. This normalisation process ensures that the metric is bounded between 0 and 1, with a score of 1 indicating a perfect ranking. That makes it easier to compare the effectiveness of different recommendation systems. It is the most common metric for most use cases. Note that, similarly to DCG, it is possible to use a truncated version of NDCG, NDCG@k, where only top k relevant items contribute to the calculation of NDCG.

4.1.3 Ranking algorithms

Several algorithms have been proposed to address the learning-to-rank task, each with its unique approach and advantages. This section will explore three prominent algorithms: RankNet, LambdaRank, and LambdaMART. These algorithms were initially developed by Christopher Burges at Microsoft and have gained considerable popularity, playing a crucial role in advancing the state-of-the-art in the ranking domain [50].

RankNet

For a given query, each possible pair of distinct items I_i and I_j with feature vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ respectively is presented to model f with parameters θ, which computes scores $s_i = f(\mathbf{x}_i)$ and $s_j = f(\mathbf{x}_j)$. Let $I_i \rhd I_j$ denote that item I_i is ranked higher than item I_j. Thus, the two output scores s_i and s_j are mapped to a probability P_{ij} that I_i is ranked higher than I_j via a sigmoid function:

$$P_{ij} = P(I_i \rhd I_j) \equiv \frac{1}{1 + e^{-\sigma(s_i - s_j)}}, \tag{4.12}$$

where σ determines the shape of the sigmoid. Let $\bar{P}_{ij}$ be the true probability that item I_i should be ranked higher than item I_j, defined as

$$\bar{P}_{ij} = \begin{cases} 1 & \text{if } I_i \rhd I_j, \\ 0 & \text{otherwise.} \end{cases} \tag{4.13}$$

To express the difference between the predicted and true probability, a cross-entropy loss function in a form

$$\mathcal{L}(I_i, I_j) = -\bar{P}_{ij} \log P_{ij} - (1 - \bar{P}_{ij}) \log(1 - P_{ij}) \tag{4.14}$$

is employed. To find the optimal model with a minimal loss, RankNet utilises a gradient descent technique, similarly to what we saw in the case of gradient boosting decision trees

in Section 4.1.1. Specifically, the gradient of the loss function $\mathcal{L}(I_i, I_j)$ with respect to the model parameters is used to update the parameters of the model as follows:

$$\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}(I_i, I_j)}{\partial \theta}, \tag{4.15}$$

where η is a parameter representing the learning rate. After the parameters of the model are updated, the optimisation process is repeated until the loss function converges to a minimum. Note that RankNet was introduced with a neural network as its function class $\mathcal{F}$. However, the underlying model is not restricted to neural networks; any model with a differentiable function of the model parameters may be used.

With minimising the loss function $\mathcal{L}$, the objective of RankNet is to minimise the number of *pairwise errors* in the ranking. A pairwise error is an incorrect order between a pair of results in a ranked list, where a lower-rated result is ranked above a higher-rated result. That is not suitable in most applications because the loss function treats the errors equally regardless of whether two items are wrongly ranked at the top or bottom of a query. However, the top of the query is crucial for most use cases. Therefore, we should mainly optimise the order of items in the first few positions. [51] proposed a possible solution to this issue in a method called LambdaRank.

LambdaRank

First, it was noticed that, when training the RankNet, only the gradients of the loss with respect to the scores are required to employ the gradient descent method, not the actual loss [51]. To obtain the desired gradients, we re-use the gradient of the loss with respect to the model parameters and rewrite it in the following way:

$$\frac{\partial \mathcal{L}(I_i, I_j)}{\partial \theta} = \frac{\partial \mathcal{L}(I_i, I_j)}{\partial s_i} \frac{\partial s_i}{\partial \theta} + \frac{\partial \mathcal{L}(I_i, I_j)}{\partial s_j} \frac{\partial s_j}{\partial \theta} = \lambda_{ij} \left(\frac{\partial s_i}{\partial \theta} - \frac{\partial s_j}{\partial \theta} \right), \tag{4.16}$$

where λ_{ij} describes the desired difference of scores for the pair of items I_i and I_j. By summing λ_{ij} over all possible sets of pairs where item I_i is a member in query q, we obtain λ_i. This λ_i corresponds to the gradient of the loss with respect to the obtained score. It can be visualised as an arrow attached to the item in the ranked list, the direction of which indicates the direction we want the item to move, and the length of which indicates by how much.

Additionally, in [51], the authors discovered a critical limitation when employing the λ gradients: their lack of positional awareness. A solution proposed to solve this issue involves scaling λ_{ij} by the change in NDCG (which is a positionally-aware ranking metric) obtained by swapping the documents:

$$\lambda_{ij} = \frac{\partial \mathcal{L}(I_i, I_j)}{\partial s_i} = \frac{-\sigma}{1 + e^{\sigma(s_i - s_j)}} |\Delta_{\text{NDCG}}|, \tag{4.17}$$

where $|\Delta_{\text{NDCG}}|$ indicates the size of the change in NDCG given by swapping the rank positions of item I_i and I_j while leaving the rank positions of all other items unchanged. The overall LambdaRank's fundamental concept is to utilise these modified gradients to train a RankNet instead of deriving them from a loss function.

LambdaMART

LambdaMART is a method inspired by LambdaRank, but based on a function class of models called multiple additive regression trees (MART). Whereas MART employs GBDT for making predictions, LambdaMART employs GBDT using a cost function based on LambdaRank for ranking tasks. As mentioned in Section 4.1.1, gradient information alone is sufficient for constructing gradient boosting models. Therefore, LamdaMART benefits from this methodology enabled by GBDT while simultaneously replacing the original gradients with λ gradients introduced in LamdbaRank. This forms the fundamental concept behind LambdaMART. Experimental datasets have shown that LambdaMART outperforms both LambdaRank and the original RankNet [50].

4.2 Dataset collection

As in every supervised machine learning problem, a labelled dataset is required to train the model. Assuming that we specifically want to develop a ranking model, we work with a dataset that comprises queries of items and a ground truth ordering of the items in queries. For a query q we have n items $\{I_1, \ldots, I_n\}$ to be ranked by their relevance. Each query-item pair (q, I_i) is represented by a numerical vector, called a *feature vector*, $\mathbf{x}_i = \phi(q, I_i) \in \mathbb{R}^m$, where m is the number of features describing the query-item pair. A relevance score r_i that says how relevant an item is to query q is assigned to each item I_i . The final training dataset is defined as $\mathcal{D} = \{(\mathbf{x}_i^q, r_i^q)\}$ for $q = 1, \ldots, Q$, $i = 1, \ldots, n_q$, where x_i^q is i-th feature vector in query q and r_i^q is a relevance score of i-th item in query q. Note that, in general, queries can be of variable length.

In learning-to-rank tasks, the dataset's structure is important. Apart from the features defining each item in the dataset and the relevance score representing the desired outcome, queries are part of the input. They play a crucial role in the learning process as they represent the context in which the items are ranked. It means that the same set of items can have a different ranking depending on the context defined by the query. In our case of ranking the pricing problems, the query refers to the current state of the restricted master problem (i.e., the values of primal and dual variables). Each item represents one of the pricing problems that can be solved. The relevance score is obtained from the reduced cost the pricing problem can achieve compared to other available pricing problems in the same query. Note that the query length is fixed in every instance to the total number of OR blocks but varies for different instances.

4.2.1 Features

The extracted feature vectors represent the characteristics of individual pricing problems to be solved at a given iteration of column generation in a given node of the branching tree. Using the features, we want to describe the pricing problem so that a ranking model can compare the usefulness of pricing problems given the context of the current restricted master problem. However, sorting the pricing problems according to the reduced cost is not a straightforward task as it is influenced by the overall structure of the problem (number

Table 4.1: Description of features extracted for a pricing problem of block b.

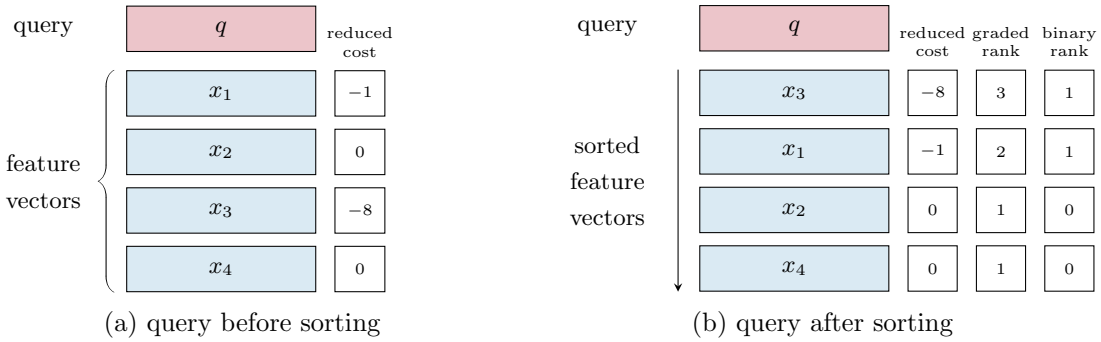
Feature	Description	Type			
		count	average	histogram	value
overall days	days in the instance	✓			
overall patients	patients in the instance	✓			
overall surgeons	surgeons in the instance	✓			
involved patterns[1]	patterns involved in block b	✓			
involved patients[1]	patients involved in block b	✓			
involved surgeons[1]	surgeons involved in block b	✓			
μ	variable μ_b extracted from RMP solution				✓
block load[1]	sum of surgery durations of patients involved in block b				✓
schedule integral	Is the current RMP solution integral in terms of block b (i.e., yes/no, 1/0)?				✓
clinical priority[1]	clinical priority of patients involved in block b		✓	✓	
waiting days[1]	waiting days of patients involved in block b		✓	✓	
surgery time[1]	surgery time of patients involved in block b		✓	✓	
setup time[1]	setup time of surgeries of patients' involved in block b		✓	✓	
patient branching 0	number of branching constraints forbidding any patient to be assigned to block b	✓			
patient branching 1	number of branching constraints assigning any patient to block b	✓			
surgeon branching 0	number of branching constraints forbidding any surgeon to be assigned to the day of block b	✓			
surgeon branching 1	number of branching constraints assigning any surgeon to the day of block b	✓			

[1] Feature is measured over the set of patterns/patients/surgeons with full involvement, partial involvement and possible involvement.

of patients, number of available surgeons, overtime of blocks, etc.), and the current solution of the master problem (optimal values of the master problem's dual variables are used as parameters in the pricing problems). Hence, no unique set of properties indicates which pricing problem will be useful.

Moreover, the definition of features is complicated because the number of variables, constraints and parameters describing the pricing problem varies according to the provided instance. However, the traditional machine learning models, therefore also the ranking models, expect the dimensionality of the input data to be always the same. As we work with instances and therefore pricing problems of arbitrary size, it is not evident at first sight how to condense the input into a fixed-size features.

We are dealing with a complex combinatorial problem with no clear way to define the features to learn from, such as when working with images or text. This fact and the issues listed above lead us to develop hand-crafted features based on expertise in the field and domain knowledge of which parameters could affect the solution of pricing problems. As a result, in Table 4.1, we propose a set of features specifically designed for our task. In total, there are 17 types of features defined. We propose four ways how to measure them: by counting



(a) query before sorting (b) query after sorting

Figure 4.1: The process of assigning binary and graded relevance scores (ranks) to a set of four pricing problems with feature vectors $x_1, \ldots, x_4$ in query q based on obtained reduced costs.

the elements (count), creating an average over the elements (average), making a histogram with a specified number of bins over the elements (histogram) or extracting the value of the feature (value). In addition, some of the features are measured over the set of patterns/patients/surgeons with three possible types of involvements in the current solution of RMP: (i) full involvement (pattern/patient/surgeon is fully assigned to block b in the current solution of RMP, i.e., the value of the corresponding variable is 1), (ii) partial involvement (pattern/patient/surgeon is partially assigned to block b in the current solution of RMP, i.e., the value of the corresponding variable is between 0 and 1), (iii) possible involvement (pattern/patient/surgeon is not assigned to block b in the current solution of RMP but could be, i.e. the value of the corresponding variable is 0). For example, when extracting the value of the clinical priority feature for block b, we first list the fully assigned patients to the block and then measure the average value and histogram over their clinical priorities. The same can be done for the partially and possibly assigned patients to the same block.

4.2.2 Relevance scores

Apart from the features, we must define relevance scores the ranking algorithm can learn from. Each pricing problem obtains a relevance score relative to other pricing problems in the same query. The score is acquired by solving all the pricing problems in one query, observing the reduced cost they produce and sorting them from the most relevant (i.e., pricing problem with the highest negative reduced cost) to the least relevant (i.e., pricing problem with the lowest negative reduced cost). This order is typically induced by giving either a binary or graded judgement. When employing binary ranking, 1 is assigned to all pricing problems that yield a negative reduced cost, while 0 is assigned to the remaining ones. Conversely, when using graded ranking, pricing problems are initially sorted based on their reduced cost and subsequently assigned increasing values starting from the lowest-ordered problem to the highest-ordered one. In this manner, the maximum achievable rank corresponds to the total number of pricing problems in the query. The process of assigning relevance scores to the pricing problems is illustrated in Figure 4.1.

4.3 Solution methodology

First, a dataset $\mathcal{D}$ is collected. The dataset consists of features (described in Section 4.2.1) and relevance scores (described in Section 4.2.2). Since we also need the ground truth relevance scores (i.e., we need to know the objective value acquired for the pricing problem), the idea for obtaining them is to solve a set of prepared instances. This means collecting all the proceeded RMP iterations as a set of queries, retrieving all the candidate pricing problems with their features and solving them to obtain the relevance scores. We build on the idea that by collecting a large number of pricing problems associated with various master problems, we can infer the impact of the pricing problems on the current master problem, even when dealing with instances that have not been encountered before.

After acquiring the dataset, the ranking model, specifically LambdaMART (described in Section 4.1.3), undergoes a training phase. During the training process, the model learns a function that takes the current state of RMP and features describing the pricing problems, and produces the relevance scores. The learning is done by adjusting the internal parameters of the LambdaMART model with underlying GBDT iteratively (described in Section 4.1.1). The objective is to minimise a loss function that quantifies the difference between the predicted and true relevance scores in the training set. However, the final goal is to be able to generalise to unseen instances. For that reason, a part of the collected dataset is reserved for testing purposes to evaluate the model's performance on unseen data. The performance is measured using NDCG metric (described in Section 4.1.2).

Once the model is trained, we incorporate it into the branch-and-price procedure. At each iteration of the column generation after re-optimising the RMP, we employ the trained ranking model to predict the ranking of pricing problems for blocks $1, \ldots, B$. From the

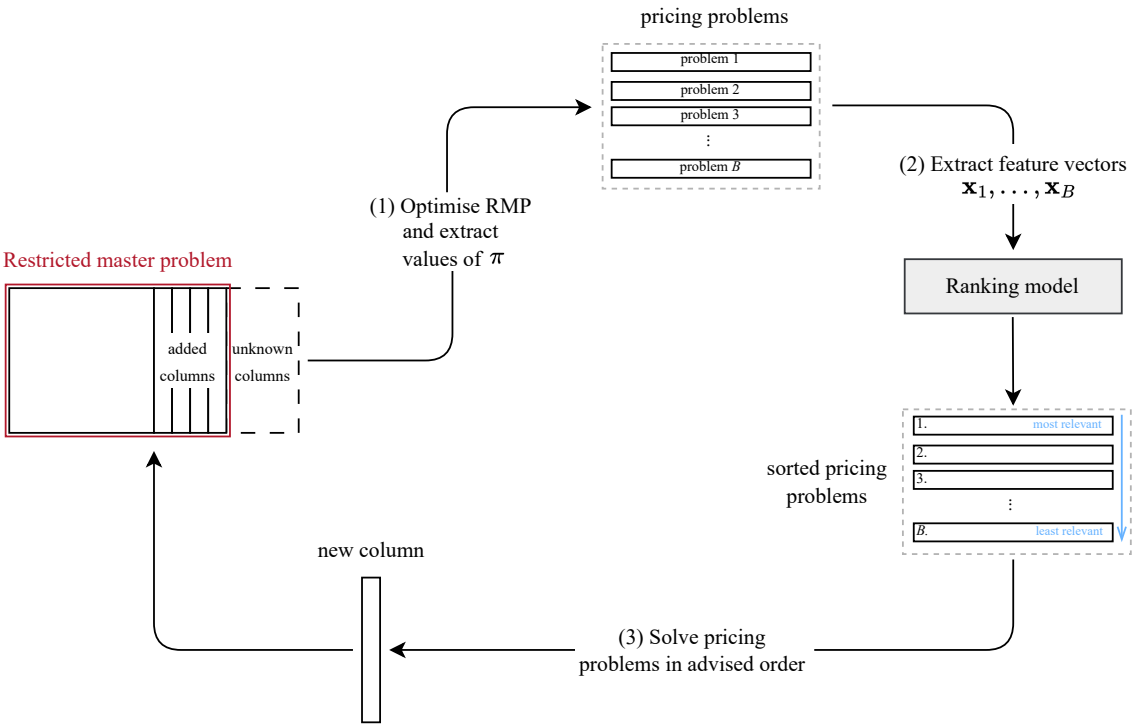


Figure 4.2: Solution process of adding new columns based on the guideline of ranking model

vectors of features $\mathbf{x}_1, \ldots, \mathbf{x}_B$ extracted for all the blocks, we compute a vector of scores $\mathbf{s} = (f(\mathbf{x}_1), \ldots, f(\mathbf{x}_B))$ and use it to compute the ordering of the pricing problems $\mathbf{z}$ from the one with the highest obtained ranking to the one with the lowest obtained ranking: $\mathbf{z} = \arg\operatorname{sort}(\mathbf{s})$. This ordering is delivered to the procedure of generating an improving pattern. It serves as a guide for exploring the space of pricing problems, indicating the order in which they should be solved. Following this order, we prioritise the most promising pricing problems, increasing the chances of finding a negative reduced cost pattern early in the process. Note that finding an optimum is guaranteed as, in the worst case, we iterate over all the pricing problems to prove that there is none with a negative reduced cost.

This process is schematically described in Figure 4.2.

Chapter 5

Experiments

In this chapter, we perform a comprehensive evaluation of the developed methods. First, in Section 5.1, we examine the instance parameters commonly employed in real-world scenarios and establish a systematic procedure for generating synthetic instances based on the parameters. Next, in Section 5.2, we compare the performance of the reference MIQP model with the baseline branch-and-price model. Section 5.3 focuses on the training and evaluation of the ML ranking model. Lastly, in Section 5.4, we compare the branch-and-price approach enhanced with the trained ranking model against the baseline branch-and-price model. We also compare multiple strategies for selecting pricing problems and the effect of adding multiple patterns at once.

In the following, we perform the experiments on two sets of instances: small and large. All experiments involving large instances were performed on a cluster with two Intel Xeon E5-2690 v4 with 14 cores operating at a frequency of 2.6 GHz with 256 GB of RAM. All experiments involving small instances were performed on a system with Apple M2 chip with 8 cores operating at a frequency of 3.49 GHz with 8 GB of RAM. All the algorithms were implemented in Python 3.9. Gurobi 10.0.0 solver was used for the LP models.

5.1 Instance generation

As a result of the complex formulation and the large number of parameters involved in the problem formulation, there is a lack of existing benchmark instances that cover all aspects of our problem. Therefore, we have developed a procedure for generating instances, which will be outlined in this section.

Traditionally, uniform distributions have been employed in scheduling to estimate parameters, enabling researchers to test the robustness of their solutions across various scenarios. However, relying solely on uniform distributions has significant drawbacks, as they may not accurately capture the complexities of real-world scenarios. While traditional scheduling algorithms remain unaffected by this limitation, machine learning models can suffer a performance drawback. To overcome this, we conduct a comprehensive hospital data survey to obtain credible parameters. The parameters predominantly draw inspiration from real plans used in the University Hospital of Hradec Králové, but adaptations were necessary to fill in the unknown gaps. For example, hospitals usually do not record the number of waiting days

of the patient or his release date. Therefore, several parameters are derived from literature or based on our domain knowledge.

The process of generating instances is as follows: the user specifies the number of ORs, blocks, surgeons, patients and a time horizon as R, B, S, P, and D, respectively. First, R operating rooms and B blocks are created. The parametrisation is generated based on Section 5.1.1 for each operating room and block. Subsequently, surgeons are created and allocated to generated blocks. The parametrisation is generated for each surgeon based on Section 5.1.2. Finally, P patients are created and allocated to generated surgeons. The parametrisation is generated for each patient based on Section 5.1.3. The instance is then finalised by establishing the objective function coefficients as described in Section 5.1.4.

5.1.1 OR blocks and rooms

Based on a real-world hospital setting, we consider a department with 1 operating room that is further split into two blocks: morning and afternoon. Considering the operating block's parameters, based on [3], the block capacity c_b is set to 240 minutes, and maximal block overtime $o_b^{\max}$ to 60 minutes. Considering the operating room's parameters, based on [3], the maximal OR overtime $o_r^{\max}$ is set to 90 minutes.

5.1.2 Surgeons

We consider a single discipline where all surgeons are equally loaded. The number of available days to conduct surgeries is generated for each surgeon. According to [18], surgeons usually attend the hospital between 3 and 5 days a week. Therefore, we assign each surgeon to 33 % of all blocks available in one week. The same time blocks are allocated to the surgeon in every week across the whole time horizon. The assignment of surgical cases to surgeons is generated uniformly at random. Parameter v of the maximal number of blocks assigned to any surgeon on any day is set to 2, as, according to Section 5.1.1, there are two blocks per day. Parameter v_s of a maximal number of patients assigned to a surgeon s in any day is set to the total number of surgeon's patients in the given time horizon.

5.1.3 Patients

We assume elective patients only. We classify patients into two clinical priority classes: normal and high-priority. It is worth noting that the priority does not stand for urgent surgeries, as we consider elective patients only. Patients might just be on the waiting list for too long, generate higher revenues for the hospital, or there is a higher medical priority. The priority can be based on release and due time [3] or can be assigned randomly. According to the Institute of Health Information and Statistics of the Czech Republic, in 2019, 17 % of the total number of surgeries in the hospitals of the Czech Republic were urgent [52]. Therefore, we randomly assign 20 % of patients to the high-priority group; the rest is assigned to the normal-priority group. The priority is reflected by the parameter q_p.

Each patient is also assigned to a group so that a setup time parameter between every two patients $t_{pp'}$ can be modelled. Introducing this parameter is motivated by the cleaning and

preparation times needed between the surgeries in real-world settings of hospitals. We define six groups, each representing one type of surgery that can be performed. Each patient is uniformly at random assigned to one of these groups. Following the findings presented in [53], we establish setup times in minutes for each pair of groups using the following matrix:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 0 & 20 & 25 & 30 & 35 & 40 \\ 20 & 0 & 20 & 25 & 30 & 35 \\ 25 & 20 & 0 & 20 & 25 & 30 \\ 30 & 25 & 20 & 0 & 20 & 25 \\ 35 & 30 & 25 & 20 & 0 & 20 \\ 40 & 35 & 30 & 25 & 20 & 0 \end{bmatrix} \end{array}.$$

The rows and columns in this matrix correspond to the group numbers (1 to 6). Note that when two patients are from the same group, no time is required to clean or prepare the operating room. Otherwise, as the distance between the group of two patients increases, the setup time also increases.

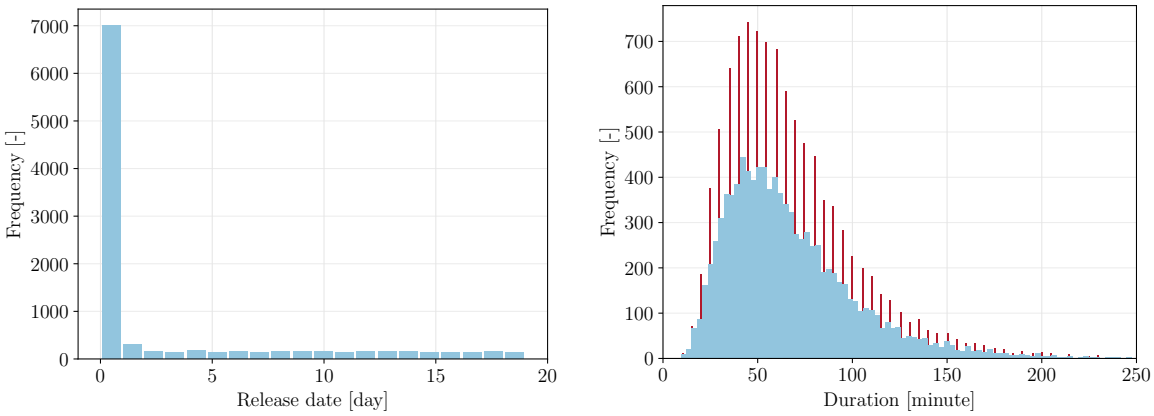
Regarding the release date a_p, we assume that 70 % of patients are randomly selected and prepared before the start of the time horizon (i.e., their release date is day 0). For the rest of the patients, we model the release date as a Poisson process in the following manner: on each day of the time horizon, we independently extract one value from the Poisson distribution with a mean equal to the ratio of unscheduled patients to the remaining time horizon. This value represents the number of patients with a release date equal to this day. Then, we randomly select this number of patients from the unscheduled patients. We iterate over the time horizon until all patients are not scheduled. The illustration of described distribution is shown in Figure 5.1. The motivation for including patients with a release date after the start of the time horizon is that some patients may have additional pre-operation procedures or are not available for personal reasons. The ratio is based on [54], where they considered a patient list with 80 initial patients rising to a total of 132 due to new arrivals.

The surgery duration of a patient can be modelled by various distributions. A log-normal distribution is assumed to copy the real-life surgery durations to a great extent [55]. Based on the mean surgery durations presented in [3], we establish a log-normal distribution with a mean of 1 hour and a standard deviation of 30 minutes. For each patient, the surgery duration parameter t_p is extracted from this distribution. The illustration of described distribution is shown in Figure 5.1.

5.1.4 Objective function

Choosing appropriate values for the coefficients m_j, $j = 1, \ldots, 5$ is critical as it significantly affects the behaviour of the optimisation process and the resulting solution. The coefficients can be determined based on various factors, such as the relative importance of the objectives or expert knowledge of the problem domain.

In [3], the authors utilised the following coefficient values: $m_1 = 30$, $m_2 = 300$, $m_3 = 15$, $m_4 = 2$. Based on these, we propose the following coefficient values for our problem: $m_1 = 10$



(a) release date a_p

(b) surgery duration t_p

Figure 5.1: Distribution of patient's parameters on a sample size 10 000.

(representing the coefficient for waiting days), $m_2 = 100$ (reflecting the coefficient for the number of unscheduled patients), $m_3 = 10$ (indicating the coefficient for the number of surgeon attendance days), $m_4 = 1$ (representing the coefficient for room overtime), and $m_5 = 1$ (reflecting the coefficient for room idle time). Our primary focus lies in minimising the number of scheduled patients. Therefore, the corresponding coefficient carries significant weight compared to the others. Additionally, we consider the number of waiting days of patients and the number of days a surgeon must be present at the hospital equally undesirable. Same holds for overtime and idle time. Furthermore, as their units are minutes, their weight is significantly lower.

It is important to note that each hospital might have different preferences. Thus, the choice of coefficients should be determined in cooperation with the hospital management.

5.2 Experiment 1: comparison of original formulation and branch-and-price formulation on large instances

We start the experiments by comparing the performance of the original formulation ILP model presented in Chapter 2 and the branch-and-price formulation presented in Chapter 3. In Chapter 3, we mentioned that ILP is impractical for large problems due to its computational complexity and scalability issues. Thus, this experiment aims to demonstrate that the branch-and-price algorithm truly outperforms ILP on generated data and is reasonable to utilise it in our task of OR scheduling, especially for large instances.

5.2.1 Settings

The experiment was conducted on instances with the following parameters: time horizon of {5,10} days, 1 OR, {30,40,50} patients, and 4 surgeons. For each combination of parameters, 3 instances were randomly generated based on the description depicted in Section 5.1, forming a dataset consisting of 18 instances. The time limit for solving one instance was set to 4 hours. A random strategy for searching for improving patterns was employed in the branch-and-price algorithm (i.e., pricing problems were solved in random order, and the first encountered

pattern with a negative reduced cost was added). One pattern per one column generation iteration was added at maximum.

5.2.2 Results

The results of the experiment are demonstrated in Table 5.1. The table is divided into two parts according to the methods. In the first part, we find the results for the ILP model, and in the second part, we find the results for the branch-and-price model. The first column of each method presents the objective value of the best solution found within the specified time limit. The second column of each method presents the runtime of the algorithm.

The results show that the ILP model performs better than the branch-and-price model for smaller instances. When considering a time horizon of 5 days, the runtime of the ILP model is, on average, approximately two times shorter than in the case of branch-and-price. For both methods, most instances can find an optimal value within the time limit; therefore, the average objective value is more or less the same. Nonetheless, when the time horizon is extended to 10 days, branch-and-price outperforms the ILP method. In the case of ILP, all but one instance reach the time limit before the optimum is found. The branch-and-price algorithm can find the optimum within the time limit for four instances. The average objective value remains more or less the same for both methods. Therefore, the branch-and-price method demonstrates its dominance for larger instances, where the ILP formulation struggles to provide optimal solutions within the given time limit. Moreover, it is crucial to understand that the branch-and-price method's performance advantage will become increasingly apparent for even larger problem sizes. As the complexity of the instances grows, the computational demands imposed on the ILP increase, failing to deliver optimal solutions within an acceptable time.

These findings emphasise the practical significance of utilising the branch-and-price algorithm for OR scheduling tasks involving large-scale instances. The algorithm's ability to handle larger instances efficiently and its competitive objective values make it a suitable choice.

5.3 Experiment 2: ML ranking model evaluation

The second experiment aims to train and evaluate a ranking model theoretically described in Chapter 4, further utilised in the branch-and-price algorithm. The section is organised similarly to the solution methodology described in Section 4.3. Accordingly, we establish the ranking model, gather data for its training, conduct the training process, and evaluate its performance. Subsequently, we analyse its performance when employed in the branch-and-price procedure. Lastly, we investigate the model's predictions on the test data.

5.3.1 Model training

First, we established the model itself. Based on the description of popular ranking models in Section 4.1.3, we decided to adopt the LamdbaMART model. A popular framework often used for learning to rank tasks and implementing the LamdbaMART model is *LigthGBM*. It is

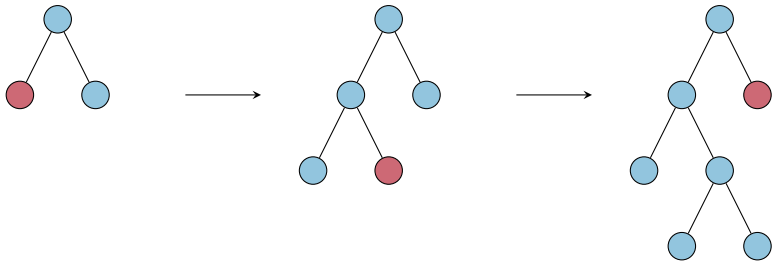
Table 5.1: Results for ILP and branch-and-price methods. Instances have following parameters: time horizon of {5,10} days, 1 OR, 4 surgeons, and {30,40,50} patients. Gray cells indicate instances that were not solved to the optimality within the given time limit.

Size	Instance	ILP		Branch-and-price	
		Value	Time [s]	Value	Time [s]
5 days	0_p30	119310	218	119310	590
	0_p40	112660	586	112660	3549
	0_p50	54690	13695	54690	10286
	1_p30	132340	49	132340	526
	1_p40	55150	4546	55150	1155
	1_p50	170115	1278	170115	2594
	2_p30	116790	192	116790	236
	2_p40	112845	388	112845	7841
	2_p50	112520	1149	112530	14403
	Average	**109602**	**2456**	**109603**	**4576**
10 days	0_p30	478315	176	478315	1522
	0_p40	400740	14401	400720	8223
	0_p50	343170	14402	343145	14405
	1_p30	419250	14401	419250	1738
	1_p40	515375	10416	515395	14408
	1_p50	515945	14403	515925	14401
	2_p30	493830	14401	493830	965
	2_p40	517150	14401	517155	14400
	2_p50	515250	14404	515275	14407
	Average	**466558**	**12378**	**466557**	**9386**

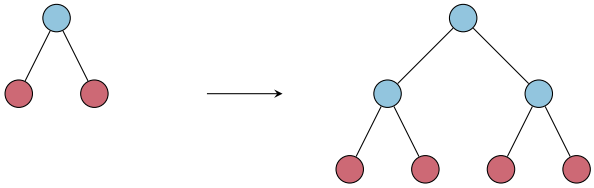
a framework developed by Microsoft that uses conventional GBDT with the addition of novel techniques. One of the key ones is its leaf-wise tree-splitting approach, instead of level-wise tree-splitting employed by other boosting algorithms. In the leaf-wise strategy, the algorithm selects the leaf node it believes will yield the largest decrease in loss at each step. Oppositely, the level-wise strategy prioritises splitting the nodes closer to the tree root growing the tree level by level. The difference is shown in Figure 5.2. Using the leaf-wise approach, the GBDT produces more complex trees resulting in a lower value of loss function than in the level-wise approach. Another key feature of LightGBM is a histogram-based algorithm used for computing gradients. It buckets continuous feature values into discrete bins, significantly reducing memory consumption and allowing it to handle large-scale datasets.

We utilised the LambdaMART model implementation within the LigthGBM framework. The used parameter settings for this model are provided in Table 5.2. Note that one of the parameters defines the objective as LambdaRank instead of LambdaMART. However, LambdaMART is considered the boosted tree version of LambdaRank. Hence, by utilising the LambdaRank objective along with LigthGBM's GBDT, we effectively obtain the LambdaMART algorithm.

Additionally, before training the model, we collected a dataset consisting of diverse instances. It was gathered by solving instances with the following parameters: time horizon of {5,10} days, 1 OR, {10,20,30} patients, and 2 surgeons. For each combination of parameters, 30



(a) leaf-wise strategy



(b) level-wise strategy

Figure 5.2: Tree growth strategies used in gradient boosted decision trees.

instances were randomly generated based on the description depicted in Section 5.1, forming a pool of 120 instances. Based on the methodology for collecting pricing problem features in queries together with ranking scores described in Section 4.3, a total number of 58 408 queries and 932 190 pricing problems were recorded, including 92 feature values and both binary and graded relevance scores.

Once the training dataset was ready, we trained the LambdaMART model. In fact, we trained two different models – one with binary and one with graded relevance scores. In that manner, the binary ranker decided whether the pricing problem was relevant, whereas the graded ranker decided to what extent the pricing problem was relevant. In both cases, the training was done on 80 % of the collected queries, and the other 20 % was used for testing purposes. The split was done randomly. In total, we trained the models for 3000 iterations, and the training process took less than 5 minutes.

To evaluate the quality of rankings produced by trained models, we employed the NDCG@k metric, where $k \in \{1, 2, 5, 10\}$, on a test part of the collected dataset. By considering the NDCG of top k ranked items only, we assess the performance of trained models in predicting

Table 5.2: Parameters used for training the ranking model.

Parameter	Description	Value
Learning rate	Boosting learning rate	0.1
Num leaves	Maximal number of leaves in one tree	31
Max depth	Maximal depth of one tree	∞
Min data	Minimal number of data in one leaf	20
Max bin	Maximal number of bins that feature values will be bucketed in	255
Num iterations	Number of boosting iterations	3000
Objective	Learning objective function	lambdarank
Metric	Metric to be evaluated on the test set	ndcg
Boosting	Type of boosting method	gbdt

k most relevant pricing problems among all the pricing problems. The results are presented in Table 5.3. We obtained NDCG@1 of 0.84 for the graded ranker, progressively increasing to NDCG@10 of 0.94. We obtained NDCG@1 of 0.93 for the binary ranker, progressively increasing to NDCG@10 of 0.96. It is apparent that both models perform reasonably well across different positions. For example, in the case of the graded learner, the NDCG@1 score of 0.84 suggests that the top-ranked pricing problem in the list has achieved 84 % of the maximum possible relevance score compared to the ideal order. Hence, the top-ranked pricing problem has a relatively high level of relevance compared to other problems. NDCG@10 score of 0.94 suggests that, on average, the top 10 pricing problems returned by the ranker are aligning by 94 % with the ideal ranking. The NDCG scores for both models increase as the position moves further down the list, indicating that the ranking quality improves as more items are considered. It is worth noting that the binary ranker generally outperforms the graded ranker in terms of NDCG scores for all positions. However, this may be caused by the binary ranker only assessing whether the PP is relevant, which is a simpler learning task than evaluating the extent to which the PP is relevant.

We have shown that both trained ranking models effectively evaluate and predict the relevance of pricing problems. They exhibit reasonable performance across different positions, as evidenced by the NDCG scores in Table 5.3. Following that, we can proceed to the experiment of incorporating machine learning models into the branch-and-price procedure and comparing the performance of both rankers on unseen data.

5.3.2 Settings

The experiment was conducted on instances with the following parameters: time horizon of {5,10} days, 1 OR, {10,20} patients, and 2 surgeons. For each combination of parameters, 5 instances were randomly generated based on the description depicted in Section 5.1, forming a dataset consisting of 20 instances. The time limit for solving one instance was set to 2 hours. A ranked strategy for searching for improving patterns was employed in the branch-and-price algorithm (i.e., pricing problems were solved in the order advised by the ranking model, and the first encountered pattern with a negative reduced cost was added). One pattern per one column generation iteration was added at maximum.

5.3.3 Results

The results of the experiment are demonstrated in Table 5.4. The table is divided into two parts according to the methods. In the first part, we find the results for the graded ranker, and in the second part, we find the results for the binary ranker. The first column of each method presents the algorithm's runtime, further split into two columns – column

Table 5.3: Resulting NDCG@k metrics for trained ranking models.

Ranker	NDCG@1	NDCG@2	NDCG@5	NDCG@10
Graded	0.84	0.87	0.93	0.94
Binary	0.93	0.93	0.95	0.96

Table 5.4: Results for branch-and-price enhanced with binary and graded ranker. Instances have following parameters: time horizon of {5,10} days, 1 OR, 2 surgeons, and {10,20} patients.

Size	Instance	Graded ranker					Binary ranker				
		Time [s]		#N	#CG	#PP	Time [s]		#N	#CG	#PP
		Total	PP				Total	PP			
5 days	0_p10	4.20	3.17	11	97	238	4.56	3.28	11	104	233
	0_p20	32.10	27.86	27	181	480	20.37	17.73	19	116	327
	1_p10	0.82	0.58	5	32	93	1.62	1.20	5	39	106
	1_p20	203.01	174.11	115	914	2508	230.02	197.07	117	939	2569
	2_p10	1.25	0.89	5	44	116	1.52	1.07	5	50	125
	2_p20	25.10	22.52	9	92	201	21.11	18.60	9	95	205
	3_p10	6.89	5.85	11	97	214	7.11	6.18	11	80	208
	3_p20	11.43	9.68	9	73	172	12.12	10.36	9	75	188
	4_p10	1.96	1.39	9	67	172	2.08	1.53	9	61	165
	4_p20	42.44	34.07	29	267	718	57.96	46.74	41	357	1001
	Avg	**32.92**	**28.01**	**23**	**186**	**491**	**35.85**	**30.38**	**24**	**192**	**513**
10 days	0_p10	5.09	3.16	9	82	294	6.30	4.31	9	73	298
	0_p20	22.94	13.49	13	144	528	31.86	19.46	13	162	565
	1_p10	3.06	1.83	9	57	288	2.79	1.73	9	55	275
	1_p20	32.86	20.79	17	177	726	33.69	21.51	17	175	690
	2_p10	8.57	5.18	31	144	895	7.68	4.74	31	132	832
	2_p20	107.97	63.19	21	526	1229	120.73	74.89	21	503	1221
	3_p10	10.23	6.83	19	120	575	8.83	5.85	19	108	538
	3_p20	302.85	237.75	25	596	1711	327.43	260.95	25	576	1699
	4_p10	12.59	7.95	51	198	1348	8.12	5.23	37	118	922
	4_p20	61.53	43.23	15	242	890	61.22	44.19	15	228	864
	Avg	**56.77**	**40.34**	**21**	**229**	**848**	**60.86**	**44.28**	**20**	**213**	**790**

'Total' measures the overall runtime, and column 'PP' measures the runtime spent on solving pricing problems. Three more columns are listed in the table beside the runtime. Column '#N' indicates the number of nodes searched in the branching tree, column '#CG' indicates the number of column generation iterations solved, and column '#PP' indicates the number of pricing problems solved.

When comparing the graded and binary rankers, concerning the number of nodes in the branching tree, column generation iterations, and pricing problems solved, both rankers perform more or less the same. For both rankers, the number of column generation iterations and solved pricing problems increases when the time horizon increases. The number of nodes in the branching tree slightly decreases. In terms of runtime, the graded ranker performs significantly better than the binary ranker. In the case of both shorter and longer time horizons, the difference in time is around 10 %. Therefore, we consider the graded ranker as the model we continue the experiments with.

Furthermore, to observe the predictions made by the graded ranker, we employ the *shapley additive explanations (SHAP)* technique. This method, introduced in [56], enables to explain individual predictions by leveraging the concept of *Shapley value*. In coalitional game theory,

the Shapley value determines each player's contribution to a collaborative work. Similarly, SHAP employs this concept, where each feature within the input feature vector acts as a player in a coalition, and SHAP quantifies the contribution of each feature towards the model's predictions [56].

Figure 5.3 represents the impact of the most important features on the model output. The position on the y-axis indicates features ordered according to their importance while the position on the x-axis represents the obtained SHAP value. Each point corresponds to a Shapley value for one feature in a particular item of the dataset. The colour indicates the feature's value from low (blue) to high (red). Overlapping points are jittered in the y-axis direction to better understand the Shapley value's spread. A dot on the left side of the axis indicates a negative influence on the prediction, whereas a dot on the right side represents a positive influence. The distance between the dot and the axis's central point signifies the

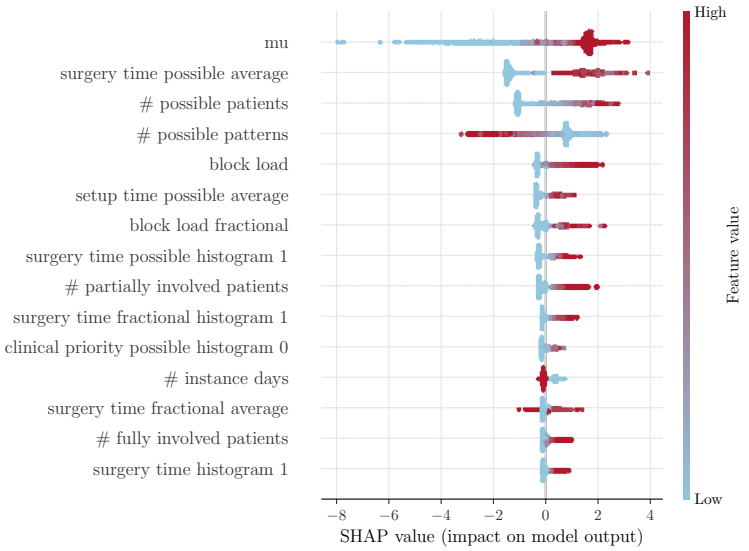


Figure 5.3: SHAP plot for graded ranker indicating the involvement of the most significant features in making predictions.

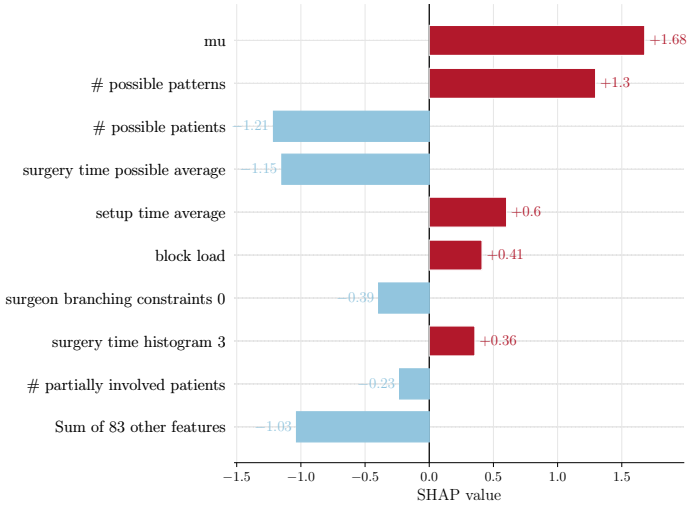


Figure 5.4: SHAP plot for graded ranker indicating how the features participate in making a single prediction.

magnitude of the effect, with greater distances indicating a stronger impact of the feature on the prediction.

By examining the plot, it is apparent that among the most influencing features is the surgery duration of patients that might be involved in the block, μ_b variable, and the number of patients and patterns that might be assigned to the block. Also, the block load plays a significant role. In most features, it holds that the higher the feature value is, the more positive impact it has on the relevance. For example, a high value of μ_b, simply from the formulation of the pricing problem, often results in a pattern with a negative reduced cost. An interesting observation is that the relevance diminishes as the number of possible patterns for a block increases. This could be attributed to the fact that when numerous patterns have already been generated for a particular block, the likelihood of the pricing problem producing a new improving pattern becomes small.

Additionally, Figure 5.4 indicates an example of how each of the features participated in making a single prediction. Again, the higher the SHAP value, the more positive impact on the relevance. For the presented pricing problem, it might be assumed that there was a relatively small number of patterns generated for the block (SHAP value is very high), increasing the relevance of the pricing problem, but, on the other hand, the number of patients that might be assigned to the block was low (SHAP value is very low), decreasing the relevance of the pricing problem.

5.4 Experiment 3: comparison of standard branch-and-price and ML-boosted branch-and-price

The final experiment compares the standard branch-and-price approach and the improved version incorporating the ranking model trained in Section 5.3. The objective is to determine if the ranking model can improve the branch-and-price algorithm in terms of computation time and search-space efficiency. Several strategies for obtaining new columns, such as random selection and prioritising high-value patterns, are assessed to evaluate the effectiveness of the enhanced approach. Furthermore, the impact of adding multiple columns at once is observed.

5.4.1 Settings

The experiment was conducted on instances with the following parameters: time horizon of {5,10} days, 1 OR, {10,20} patients, and 2 surgeons. For each combination of parameters, 5 instances were randomly generated based on the description depicted in Section 5.1, forming a dataset consisting of 20 instances. The time limit for solving one instance was set to 2 hours. Multiple strategies for searching for new columns were utilised:

- "k improving" strategy – pricing problems are randomly sorted; the algorithm searches for the first k improving patterns; these patterns are subsequently added to the RMP
- "k fixed" strategy – the pricing problems are randomly sorted; the algorithm solves first k patterns; all improving patterns among the solved ones are subsequently added to the RMP

- "k best" strategy – pricing problems are randomly sorted; the algorithm solves all of them; k patterns with the highest negative reduced cost are added to the RMP
- "k ML improving" strategy – pricing problems are sorted in the order advised by the ranking model; the algorithm searches for the first k improving patterns; these patterns are subsequently added to the RMP
- "k ML fixed" strategy – the pricing problems are sorted in the order advised by the ranking model; the algorithm solves first k patterns; all improving patterns among the solved ones are subsequently added to the RMP,

where $k \in \{1, 3, 5\}$ is the maximum number of columns added at once to the RMP. In all strategies, if it is impossible to find k improving patterns, all found improving patterns are added to the RMP.

5.4.2 Results

The results of the experiment are demonstrated in Table 5.5. The presented numbers are the averages over all instances in the dataset. The first three columns, 'Size', 'Patterns' and 'Strategy', represent the considered time horizon, number of patterns to be added, and strategy for obtaining new columns, respectively. The next column presents the algorithm's runtime, further split into three columns – column 'Total' measures the overall runtime, column 'PP' measures the runtime spent on solving pricing problems and column 'ML' measures the runtime spent on making the predictions with the ranking model. Four more columns are listed in the table beside the runtime. Column '#N' indicates the number of nodes searched in the branching tree, column '#CG' indicates the number of column generation iterations solved, column '#PP' indicates the number of pricing problems solved, and column 'PP reduced' indicates the percentage by which the ML-enhanced version of branch-and-price reduces the number of pricing problems solved compared to the best-performing strategy of branch-and-price without ranker.

The first observation from the results is that the ML-boosted branch-and-price approach consistently reduces the number of solved pricing problems compared to the standard branch-and-price approach. As indicated in the last column of the table, both in the shorter and longer time horizon, savings are, in most cases, over 30 %. An exception occurs for instances with a time horizon of five days when more patterns are added to the RMP. In this case, with a total of 10 pricing problems to be searched, finding five improvements becomes rare, resulting in an almost complete exploration of the search space. Therefore, the ML-enhanced approach reaches its limits in reducing the number of solved pricing problems when the number of patterns to be added exceeds the available opportunities for significant improvements. In other cases, the results are considerable, and the ranking model provides valuable guidance in selecting promising patterns.

It is also important to note that there is always a minimum number of patterns that must be solved, regardless of the approach employed. In each column generation iteration, at best, one pricing problem has to be solved to find an improving pattern. Additionally, in each node, at least once in the last column generation iteration, all possible pricing problems have to be solved to prove that there is no improving pattern, and we solved the RMP to optimality. For example, when the time horizon is 5 days and "ML fixed" strategy is

employed, 22 nodes and 179 column generation iterations are utilised. It means that at least $178 + 10 \cdot 22 = 398$ pricing problems have to be solved in the optimal case. In reality, we solved a total of 480 problems. This outcome indicates that the algorithm effectively navigated the problem space, minimising solving irrelevant pricing problems and reaching almost the minimal number of them.

Another observation is that according to the column 'PP' in the 'Time' section, the majority of the run time is spent on solving pricing problems. Therefore, utilising a machine learning guide to reduce the number of pricing problems should significantly improve efficiency. However, although the ML-enhanced approach significantly reduces the number of solved pricing problems, the overall runtime is not proportionately reduced. Nevertheless, the ML component itself only constitutes a small portion of time spent on solving subproblems, as the column 'ML' in the 'Time' section indicates. Hence, the additional time spent on solving pricing problems can be most likely attributed to the fact that the ML-enhanced approach tends to recommend more complex pricing problems that are likely to yield significant improvements in the objective value. It is worth noting that for a broader time horizon or multiple operating rooms, the number of pricing problems to solve will increase. Thus, we can expect that in such cases, time savings will increase because the chance of finding an improving pattern may decrease when randomly searching the space, especially if there is a limited number of them.

In terms of runtime, for both the shorter and longer time horizon, the strategy of selecting 5 patterns at once is the best-found option. In the case of a time horizon of 5 days, the best-obtained result is 21.29 seconds on average, obtained by the "fixed" strategy of standard branch-and-price. In the case of a time horizon of 10 days, the best-obtained result is 26.22 seconds on average, obtained by the "ML improving" strategy of machine learning enhanced branch-and-price. In the later, the runtime is reduced by 17 % compared to the best runtime obtained by traditional branch-and-price on the same instances.

Furthermore, the ML-boosted branch-and-price approach consistently saves time compared to the "best" strategy. This time-saving effect is particularly prominent for larger instances when a smaller number of patterns to be added is considered. This indicates that the ranking model provides valuable guidance in selecting promising patterns for column generation. However, surprisingly, even a random strategy for obtaining new columns outperforms the best choice strategy in terms of runtime.

Table 5.5: Results for baseline branch-and-price and branch-and-price enhanced with graded ranker. Instances have following parameters: time horizon of {5,10} days, 1 OR, 2 surgeons, and {10,20} patients. Averages over instances are presented in the table.

Size	Patterns	Strategy	Time [s]			#N	#CG	#PP	PP reduced [%]
			Total	PP	ML				
5 days	1	improving	29.92	24.43		30.00	213.70	922.00	
		fixed	30.57	24.93		30.00	213.70	922.00	
		best	54.83	49.92		26.60	202.80	2028.00	
		ML improving	28.41	24.27	0.90	22.20	179.40	479.50	48
		ML fixed	29.98	25.39	1.02	22.20	179.40	479.50	48
	3	improving	27.10	23.83		24.20	107.00	974.00	
		fixed	28.89	24.34		28.20	173.20	908.60	
		best	21.58	19.24		20.60	92.60	926.00	
		ML improving	27.91	24.71	0.62	23.60	110.20	859.10	5
		ML fixed	30.65	26.31	0.80	28.20	140.10	639.10	30
	5	improving	24.48	21.67		21.00	88.30	874.70	
		fixed	21.29	18.47		21.80	130.00	824.40	
		best	30.38	26.98		26.40	109.00	1090.00	
		ML improving	28.98	25.83	0.59	25.40	109.40	1056.10	0
		ML fixed	27.39	24.30	0.63	25.40	109.40	674.00	18
10 days	1	improving	36.55	23.18		23.20	211.30	1286.50	
		fixed	35.79	22.44		23.20	211.30	1286.50	
		best	83.06	71.14		19.00	177.10	3542.00	
		ML improving	52.77	37.83	2.05	21.00	222.60	841.10	35
		ML fixed	55.95	39.35	2.02	21.00	222.60	841.10	35
	3	improving	32.68	25.32		21.80	102.10	1417.60	
		fixed	36.40	22.85		24.40	190.70	1343.50	
		best	51.05	41.24		19.80	105.00	2100.00	
		ML improving	31.24	23.59	0.86	20.80	96.80	917.10	32
		ML fixed	40.96	30.12	1.22	25.40	134.80	919.40	32
	5	improving	31.66	24.42		26.00	84.70	1512.80	
		fixed	38.47	26.51		23.20	170.00	1422.70	
		best	35.94	28.71		19.40	75.90	1518.00	
		ML improving	26.22	20.35	0.67	21.00	73.50	1010.30	29
		ML fixed	32.34	24.03	0.78	20.80	85.20	758.50	47

Chapter 6

Conclusion

In this thesis, we addressed the problem of surgery planning and designed an algorithm that assigns patients to operating rooms. Our work made several contributions to the field of surgery scheduling and machine learning.

Firstly, we formulated the OR scheduling problem as a MIQP with an objective function simultaneously incorporating multiple factors. By taking into account sequence-dependent setup times, we addressed both the advance scheduling problem and the allocation scheduling problem in an integrated manner.

Given the complexity of the problem, it was crucial to design a method that could reduce the computational time required to solve it. Therefore, we developed a branch-and-price algorithm that leveraged the power of column generation and branch-and-bound methods to handle many variables efficiently. Moreover, several improvements were made to the branch-and-price algorithm leading to the ability to demonstrate its benefits over the original MIQP formulation on larger instances.

One of the main contributions of this work is the design of a novel application of machine learning in the branch-and-price algorithm. We introduced a machine learning-based ranker that guides the search for new variables in the column generation process. The ranker analyses the structure of the instance being solved and ranks the pricing problems based on the expected amount of negative reduced cost they can obtain.

To evaluate the effectiveness of the developed algorithm, we constructed a synthetic data generation method that accurately reflected real-world scenarios. This allowed us to test our algorithm on a wide range of instances. The experimental evaluation showed that our algorithm, enhanced with machine learning, significantly outperformed the baseline method. We achieved a reduction of up to 48 % in the number of solved pricing problems and up to 17 % in the overall computation time.

Results have demonstrated the efficiency of our solution, making it applicable to other optimisation problems where the computational time is primarily spent on solving pricing problems. Moreover, the results have demonstrated significant promise in merging the fields of machine learning and combinatorial optimisation. Although the techniques employed in this thesis are still in the early stages of development, combining the strengths of both disciplines has opened up new avenues for innovation in solving complex optimisation problems.

6.1 Future work

While this thesis has made significant strides in improving OR scheduling through optimisation and machine learning, there are several avenues for future research and development.

So far, we have designed the task of ranking pricing problems so the amount of negative reduced cost was considered. However, a pattern with negative reduced cost does not guarantee an improvement in the objective value of RMP. To address this limitation, we can move from ranking pricing problems according to the reduced cost to ranking pricing problems according to an improvement in the objective value of RMP they yield. Nonetheless, although these patterns might guide the algorithm towards an optimal RMP solution, there is still a possibility that they may not align well with the binary decisions we encounter during branching. In such cases, generated patterns could become irrelevant and be eliminated. Therefore, we could move the task even further by asking how likely the pattern is to be present in the final integer solution.

The results have shown that the ranking model provides valuable guidance in selecting promising patterns in medium-sized instances. However, further investigation on larger instances should be performed to provide additional insights into the effectiveness of the ranking model. When utilising more rooms and enlarging the time horizon, the number of OR blocks grows. In such a setting, the effect of machine learning is expected to be even more significant as the probability of discovering an improving pattern randomly decreases, particularly if the number of improving patterns is limited.

In addition to incorporating larger instances, it is worth exploring additional strategies for searching for improving patterns. We performed the final experiments only when 1, 3 and 5 patterns were added simultaneously. One potential approach is incorporating more selected patterns, such as 10, 15, or 20. This variation in the number of patterns allows for broader solution-space exploration. Furthermore, it would be beneficial to experiment with a dynamically changing number of patterns added. For instance, the current depth of the search could be considered. By adjusting the number of added patterns based on depth, the search process could add more patterns at once early in the tree, where the potential for generating many improving patterns is high, while saving time on solving pricing problems in deeper areas of the tree, where improving patterns are generated rarely.

Lastly, improvements may also be considered regarding the developed machine learning method. The results have shown that the training of the model and its inference can be done in a very short time. Therefore, an idea worth testing is an online utilisation of the ranking model. We may consider to incorporate the training process directly in the branching tree, learning from the instance being solved, instead of training the model on historical data. This way, we can enable the model to dynamically adjust the rankings based on the problem at hand. Additionally, we may consider utilising different types of machine learning techniques, such as graph neural networks, which have shown promising results in various problems in combinatorial optimisation.

Bibliography

[1] Z. Abdelrasol, N. Harraz, and A. Eltawil, "Operating room scheduling problems: A survey and a proposed solution framework", in *Transactions on Engineering Technologies*, Springer Netherlands, 2014, pp. 717–731. DOI: 10.1007/978-94-017-9115-1_52. [Online]. Available: https://doi.org/10.1007/978-94-017-9115-1_52.

[2] F. Guerriero and R. Guido, "Operational research in the management of the operating theatre: A survey", *Health Care Management Science*, vol. 14, no. 1, pp. 89–114, Nov. 2010. DOI: 10.1007/s10729-010-9143-6. [Online]. Available: https://doi.org/10.1007/s10729-010-9143-6.

[3] M. A. Kamran, B. Karimi, and N. Dellaert, "Uncertainty in advance scheduling problem in operating room planning", *Computers & Industrial Engineering*, vol. 126, pp. 252–268, 2018, ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2018.09.030. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360835218304467.

[4] B. Cardoen, E. Demeulemeester, and J. Beliën, "Operating room planning and scheduling: A literature review", *European Journal of Operational Research*, vol. 201, no. 3, pp. 921–932, Mar. 2010. DOI: 10.1016/j.ejor.2009.04.011. [Online]. Available: https://doi.org/10.1016/j.ejor.2009.04.011.

[5] H. Fei, N. Meskens, and C. Chu, "A planning and scheduling problem for an operating theatre using an open scheduling strategy", *Computers & Industrial Engineering*, vol. 58, no. 2, pp. 221–230, Mar. 2010. DOI: 10.1016/j.cie.2009.02.012. [Online]. Available: https://doi.org/10.1016/j.cie.2009.02.012.

[6] S. Zhu, W. Fan, S. Yang, J. Pei, and P. M. Pardalos, "Operating room planning and surgical case scheduling: A review of literature", *Journal of Combinatorial Optimization*, vol. 37, no. 3, pp. 757–805, Jul. 2018. DOI: 10.1007/s10878-018-0322-6. [Online]. Available: https://doi.org/10.1007/s10878-018-0322-6.

[7] S. H. H. Doulabi, L.-M. Rousseau, and G. Pesant, "A constraint-programming-based branch-and-price-and-cut approach for operating room planning and scheduling", *INFORMS Journal on Computing*, vol. 28, no. 3, pp. 432–448, Jul. 2016. DOI: 10.1287/ijoc.2015.0686. [Online]. Available: https://doi.org/10.1287/ijoc.2015.0686.

[8] M. Lamiri, X. Xie, and S. Zhang, "Column generation approach to operating theater planning with elective and emergency patients", *IIE Transactions*, vol. 40, no. 9, pp. 838–852, Jul. 2008. DOI: 10.1080/07408170802165831. [Online]. Available: https://doi.org/10.1080/07408170802165831.

[9] A. Riise, C. Mannino, and E. K. Burke, "Modelling and solving generalised operational surgery scheduling problems", *Computers & Operations Research*, vol. 66, pp. 1–11, Feb. 2016. DOI: 10.1016/j.cor.2015.07.003. [Online]. Available: https://doi.org/10.1016/j.cor.2015.07.003.

[10] H. Fei, C. Chu, and N. Meskens, "Solving a tactical operating room planning problem by a column-generation-based heuristic procedure with four criteria", *Annals of Operations Research*, vol. 166, no. 1, pp. 91–108, Aug. 2008. DOI: 10.1007/s10479-008-0413-3. [Online]. Available: https://doi.org/10.1007/s10479-008-0413-3.

[11] B. Roland, C. D. Martinelly, F. Riane, and Y. Pochet, "Scheduling an operating theatre under human resource constraints", *Computers & Industrial Engineering*, vol. 58, no. 2, pp. 212–220, Mar. 2010. DOI: 10.1016/j.cie.2009.01.005. [Online]. Available: https://doi.org/10.1016/j.cie.2009.01.005.

[12] R. Guido and D. Conforti, "A hybrid genetic approach for solving an integrated multi-objective operating room planning and scheduling problem", *Computers & Operations Research*, vol. 87, pp. 270–282, Nov. 2017. DOI: 10.1016/j.cor.2016.11.009. [Online]. Available: https://doi.org/10.1016/j.cor.2016.11.009.

[13] B. Cardoen, E. Demeulemeester, and J. Beliën, "Sequencing surgical cases in a day-care environment: An exact branch-and-price approach", *Computers & Operations Research*, vol. 36, no. 9, pp. 2660–2669, Sep. 2009. DOI: 10.1016/j.cor.2008.11.012. [Online]. Available: https://doi.org/10.1016/j.cor.2008.11.012.

[14] E. Hans, G. Wullink, M. van Houdenhoven, and G. Kazemier, "Robust surgery loading", *European Journal of Operational Research*, vol. 185, no. 3, pp. 1038–1050, Mar. 2008. DOI: 10.1016/j.ejor.2006.08.022. [Online]. Available: https://doi.org/10.1016/j.ejor.2006.08.022.

[15] I. Marques, M. E. Captivo, and M. V. Pato, "A bicriteria heuristic for an elective surgery scheduling problem", *Health Care Management Science*, vol. 18, no. 3, pp. 251–266, Oct. 2014. DOI: 10.1007/s10729-014-9305-z. [Online]. Available: https://doi.org/10.1007/s10729-014-9305-z.

[16] I. Marques, M. E. Captivo, and M. V. Pato, "An integer programming approach to elective surgery scheduling", *OR Spectrum*, vol. 34, no. 2, pp. 407–427, Dec. 2011. DOI: 10.1007/s00291-011-0279-7. [Online]. Available: https://doi.org/10.1007/s00291-011-0279-7.

[17] M. Younespour, A. Atighehchian, K. Kianfar, and E. T. Esfahani, "Using mixed integer programming and constraint programming for operating rooms scheduling with modified block strategy", *Operations Research for Health Care*, vol. 23, p. 100 220, Dec. 2019. DOI: 10.1016/j.orhc.2019.100220. [Online]. Available: https://doi.org/10.1016/j.orhc.2019.100220.

[18] H. Fei, C. Chu, N. Meskens, and A. Artiba, "Solving surgical cases assignment problem by a branch-and-price approach", *International Journal of Production Economics*, vol. 112, no. 1, pp. 96–108, Mar. 2008. DOI: 10.1016/j.ijpe.2006.08.030. [Online]. Available: https://doi.org/10.1016/j.ijpe.2006.08.030.

[19] R. Aringhieri, P. Landa, P. Soriano, E. Tànfani, and A. Testi, "A two level metaheuristic for the operating room scheduling and assignment problem", *Computers & Operations Research*, vol. 54, pp. 21–34, Feb. 2015. DOI: 10.1016/j.cor.2014.08.014. [Online]. Available: https://doi.org/10.1016/j.cor.2014.08.014.

[20] R. Bargetto, T. Garaix, and X. Xie, "A branch-and-price-and-cut algorithm for operating room scheduling under human resource constraints", *Computers & Operations Research*, vol. 152, p. 106 136, Apr. 2023. DOI: 10.1016/j.cor.2022.106136. [Online]. Available: https://doi.org/10.1016/j.cor.2022.106136.

[21] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon", *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, Apr. 2021. DOI: 10.1016/j.ejor.2020.07.063. [Online]. Available: https://doi.org/10.1016/j.ejor.2020.07.063.

[22] J. Chu, C.-H. Hsieh, Y.-N. Shih, *et al.*, "Operating room usage time estimation with machine learning models", *Healthcare*, vol. 10, no. 8, p. 1518, Aug. 2022. DOI: 10.3390/healthcare10081518. [Online]. Available: https://doi.org/10.3390/healthcare10081518.

[23] B. Abbou, O. Tal, G. Frenkel, R. Rubin, and N. Rappoport, "Optimizing operation room utilization—a prediction model", *Big Data and Cognitive Computing*, vol. 6, no. 3, p. 76, Jul. 2022. DOI: 10.3390/bdcc6030076. [Online]. Available: https://doi.org/10.3390/bdcc6030076.

[24] M. A. Bartek, R. C. Saxena, S. Solomon, *et al.*, "Improving operating room efficiency: Machine learning approach to predict case-time duration", *Journal of the American College of Surgeons*, vol. 229, no. 4, 346–354e3, Oct. 2019. DOI: 10.1016/j.jamcollsurg.2019.05.029. [Online]. Available: https://doi.org/10.1016/j.jamcollsurg.2019.05.029.

[25] L. Luo, F. Zhang, Y. Yao, R. Gong, M. Fu, and J. Xiao, "Machine learning for identification of surgeries with high risks of cancellation", *Health Informatics Journal*, vol. 26, no. 1, pp. 141–155, Dec. 2018. DOI: 10.1177/1460458218813602. [Online]. Available: https://doi.org/10.1177/1460458218813602.

[26] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi, *Exact combinatorial optimization with graph convolutional neural networks*, 2019. DOI: 10.48550/ARXIV.1906.01629. [Online]. Available: https://arxiv.org/abs/1906.01629.

[27] A. M. Alvarez, Q. Louveaux, and L. Wehenkel, "A machine learning-based approximation of strong branching", *INFORMS Journal on Computing*, vol. 29, no. 1, pp. 185–195, Jan. 2017. DOI: 10.1287/ijoc.2016.0723. [Online]. Available: https://doi.org/10.1287/ijoc.2016.0723.

[28] N. Furian, M. O'Sullivan, C. Walker, and E. Çela, "A machine learning-based branch and price algorithm for a sampled vehicle routing problem", *OR Spectrum*, vol. 43, no. 3, pp. 693–732, Jan. 2021. DOI: 10.1007/s00291-020-00615-8. [Online]. Available: https://doi.org/10.1007/s00291-020-00615-8.

[29] S. Kraul, M. Seizinger, and J. O. Brunner, "Machine learning–supported prediction of dual variables for the cutting stock problem with an application in stabilized column generation", *INFORMS Journal on Computing*, Mar. 2023. DOI: 10.1287/ijoc.2023.1277. [Online]. Available: https://doi.org/10.1287/ijoc.2023.1277.

[30] R. Václavík, A. Novák, P. Šůcha, and Z. Hanzálek, "Accelerating the branch-and-price algorithm using machine learning", *European Journal of Operational Research*, vol. 271, no. 3, pp. 1055–1069, Dec. 2018. DOI: 10.1016/j.ejor.2018.05.046. [Online]. Available: https://doi.org/10.1016/j.ejor.2018.05.046.

[31] Y. Shen, Y. Sun, X. Li, A. Eberhard, and A. Ernst, *Enhancing column generation by a machine-learning-based pricing heuristic for graph coloring*, 2021. DOI: 10.48550/ARXIV.2112.04906. [Online]. Available: https://arxiv.org/abs/2112.04906.

[32] F. Quesnel, A. Wu, G. Desaulniers, and F. Soumis, "Deep-learning-based partial pricing in a branch-and-price algorithm for personalized crew rostering", *Computers & Operations Research*, vol. 138, p. 105 554, Feb. 2022. DOI: 10.1016/j.cor.2021.105554. [Online]. Available: https://doi.org/10.1016/j.cor.2021.105554.

[33] M. Morabit, G. Desaulniers, and A. Lodi, "Machine-learning–based arc selection for constrained shortest path problems in column generation", *INFORMS Journal on Optimization*, Oct. 2022. DOI: 10.1287/ijoo.2022.0082. [Online]. Available: https://doi.org/10.1287/ijoo.2022.0082.

[34] A. Tahir, F. Quesnel, G. Desaulniers, I. E. Hallaoui, and Y. Yaakoubi, "An improved integral column generation algorithm using machine learning for aircrew pairing", *Transportation Science*, vol. 55, no. 6, pp. 1411–1429, Nov. 2021. DOI: 10.1287/trsc.2021.1084. [Online]. Available: https://doi.org/10.1287/trsc.2021.1084.

[35] S. Li, Z. Yan, and C. Wu, *Learning to delegate for large-scale vehicle routing*, 2021. DOI: 10.48550/ARXIV.2107.04139. [Online]. Available: https://arxiv.org/abs/2107.04139.

[36] H. Yuan, P. Jiang, and S. Song, *The neural-prediction based acceleration algorithm of column generation for graph-based set covering problems*, 2022. DOI: 10.48550/ARXIV.2207.01411. [Online]. Available: https://arxiv.org/abs/2207.01411.

[37] M. Morabit, G. Desaulniers, and A. Lodi, "Machine-learning–based column selection for column generation", *Transportation Science*, vol. 55, no. 4, pp. 815–831, Jul. 2021. DOI: 10.1287/trsc.2021.1045. [Online]. Available: https://doi.org/10.1287/trsc.2021.1045.

[38] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance, "Branch-and-price: Column generation for solving huge integer programs", *Operations Research*, vol. 46, no. 3, pp. 316–329, Jun. 1998. DOI: 10.1287/opre.46.3.316. [Online]. Available: https://doi.org/10.1287/opre.46.3.316.

[39] M. E. Lübbecke, *Column generation*, Jan. 2011. DOI: 10.1002/9780470400531.eorms0158. [Online]. Available: https://doi.org/10.1002/9780470400531.eorms0158.

[40] G. B. Dantzig and P. Wolfe, "The decomposition algorithm for linear programs", *Econometrica*, vol. 29, no. 4, p. 767, Oct. 1961. DOI: 10.2307/1911818. [Online]. Available: https://doi.org/10.2307/1911818.

[41] D. Feillet, "A tutorial on column generation and branch-and-price for vehicle routing problems", *4OR*, vol. 8, no. 4, pp. 407–424, Jun. 2010. DOI: 10.1007/s10288-010-0130-z. [Online]. Available: https://doi.org/10.1007/s10288-010-0130-z.

[42] B. Maenhout and M. Vanhoucke, "Branching strategies in a branch-and-price approach for a multiple objective nurse scheduling problem", *Journal of Scheduling*, vol. 13, no. 1, pp. 77–93, Mar. 2009. DOI: 10.1007/s10951-009-0108-x. [Online]. Available: https://doi.org/10.1007/s10951-009-0108-x.

[43] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms* (The MIT Press), 3rd ed. London, England: MIT Press, Jul. 2009.

[44] E. Danna and C. L. Pape, "Branch-and-price heuristics: A case study on the vehicle routing problem with time windows", in *Column Generation*, Springer US, 2005, pp. 99–129. DOI: 10.1007/0-387-25486-2_4. [Online]. Available: https://doi.org/10.1007/0-387-25486-2_4.

[45] A. Atamtürk, G. L. Nemhauser, and M. W. P. Savelsbergh, "A combined lagrangian, linear programming, and implication heuristic for large-scale set partitioning problems", *Journal of Heuristics*, vol. 1, no. 2, pp. 247–259, 1996. DOI: 10.1007/bf00127080. [Online]. Available: https://doi.org/10.1007/bf00127080.

[46] G. Desaulniers, J. Desrosiers, and M. M. Solomon, "Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems", in *Operations Research/Computer Science Interfaces Series*, Springer US, 2002, pp. 309–324. DOI: 10.1007/978-1-4615-1507-4_14. [Online]. Available: https://doi.org/10.1007/978-1-4615-1507-4_14.

[47] T.-Y. Liu, *Learning to Rank for Information Retrieval*. Springer Berlin Heidelberg, 2011. DOI: 10.1007/978-3-642-14267-3. [Online]. Available: https://doi.org/10.1007/978-3-642-14267-3.

[48] L. Katzir, G. Elidan, and R. El-Yaniv, "Net-dnf: Effective deep modeling of tabular data", in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=73WTGs96kho.

[49] J. H. Friedman, "Greedy function approximation: A gradient boosting machine.", *The Annals of Statistics*, vol. 29, no. 5, Oct. 2001. DOI: 10.1214/aos/1013203451. [Online]. Available: https://doi.org/10.1214/aos/1013203451.

[50] C. J. C. Burges, "From RankNet to LambdaRank to LambdaMART: An overview", Microsoft Research, Tech. Rep., 2010. [Online]. Available: http://research.microsoft.com/en-us/um/people/cburges/tech_reports/MSR-TR-2010-82.pdf.

[51] C. Burges, R. Ragno, and Q. Le, "Learning to rank with nonsmooth cost functions", in *Advances in Neural Information Processing Systems*, B. Schölkopf, J. Platt, and T. Hoffman, Eds., vol. 19, MIT Press, 2006. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2006/file/af44c4c56f385c43f2529f9b1b018f6a-Paper.pdf.

[52] ÚZIS, *Hospitalizovaní v nemocnicích čr 2019*. [Online]. Available: https://www.uzis.cz/index.php?pg=record&id=8357.

[53] G. Latorre-Núñez, A. Lüer-Villagra, V. Marianov, C. Obreque, F. Ramis, and L. Neriz, "Scheduling operating rooms with consideration of all resources, post anesthesia beds and emergency surgeries", *Computers & Industrial Engineering*, vol. 97, pp. 248–257, Jul. 2016. DOI: 10.1016/j.cie.2016.05.016. [Online]. Available: https://doi.org/10.1016/j.cie.2016.05.016.

[54] B. Addis, G. Carello, A. Grosso, and E. Tànfani, "Operating room scheduling and rescheduling: A rolling horizon approach", vol. 28, no. 1-2, pp. 206–232, Jan. 2015. DOI: 10.1007/s10696-015-9213-7. [Online]. Available: https://doi.org/10.1007/s10696-015-9213-7.

[55] J. H. May, D. P. Strum, and L. G. Vargas, "Fitting the lognormal distribution to surgical procedure times", *Decision Sciences*, vol. 31, no. 1, pp. 129–148, Mar. 2000. DOI: 10.1111/j.1540-5915.2000.tb00927.x. [Online]. Available: https://doi.org/10.1111/j.1540-5915.2000.tb00927.x.

[56] S. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions.", *CoRR*, vol. abs/1705.07874, 2017. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1705.html#LundbergL17.

Appendix

Contents of the attachment

```
/
├── src ..... source code
├── text ..... LaTeX source files
└── DP_Koutecka_2023.pdf ..... own text of the thesis
```