CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

MASTER'S THESIS

Antonín Novák

Methods of the efficient state space search for the Nurse Rostering Problem using branch-and-price approach

Department of Computer Science

Thesis supervisor: **Ing. Roman Václavík**

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 11. 05. 2015 .... [redacted] ....

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Science and Engineering

DIPLOMA THESIS ASSIGNMENT

Student: **Bc. Antonín Novák**

Study programme: Open Informatics
Specialisation: Artificial Intelligence

Title of Diploma Thesis: **Methods of the efficient state space search for the nurse rostering problem using branch-and-price approach**

Guidelines:

The aim of this work is to implement the exact algorithm for the nurse rostering problem. The algorithm will be based on the branch-and-price approach that will be extended by suitable methods from the machine learning domain. These methods will be used to accelerate the algorithm runtime due to gained knowledge from the previous algorithm iterations or algorithm runs.

Assignment:

1. Conduct the research about the nurse rostering problem focused on the methods for solving the problem.
2. Analyze the branch-and-price approach. Identify the parts which can be improved by machine learning methods.
3. Propose the tailor-made machine learning methods which are suitable for the identified parts.
4. Implement the algorithm based on the branch-and-price method incorporating the proposed improvements.
5. Test and compare the implemented algorithm on the benchmark instances and discuss the results.

Bibliography/Sources:

[1] E. K. Burke, T. Curtois: New approaches to nurse rostering benchmark instances. European Journal of Operational Research, 71-81, 2014.
[2] B. Maenhout, M. Vanhoucke: Branching strategies in a branch-and-price approach for a multiple objective nurse scheduling problem. Journal of Scheduling, 77 - 93, 2010.
[3] K. P. Murphy: Machine Learning: A Probabilistic Perspective. MIT Press, 2012.
[4] T. Curtois: Staff rostering benchmark data sets. http://www.cs.nott.ac.uk/~tec/NRP/, 2014.

Diploma Thesis Supervisor: Ing. Roman Václavík

Valid until the end of the summer semester of academic year 2015/2016

[illegible]

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
-4-
L.S.

[illegible]

doc. Ing. Filip Železný, Ph.D.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, March 25, 2015

Acknowledgements

I would like to thank my family for support during writing this thesis and throughout my studies. Without them this work would not be possible. Moreover, thanks belongs to my girlfriend who patiently supported me over the years and provided me a calm environment for my work. Furthermore, I would like to thank my thesis supervisor and Přemysl Šůcha for great insights and discussions. Finally, my thanks goes to my friend Tomáš Báča for his excellent comments and helpful corrections.

Abstract

This thesis introduces an exact algorithm for Nurse Rostering Problem based on general decomposition method called Branch and Price. Proposed method handles expressive set of soft constraints and is suitable for real-world personnel rostering problems, which is shown by evaluation on publicly available benchmark instances collected from real-world environments. Furthermore, we have shown, how solutions of related hard optimization problems can be used for future ones. Moreover, we studied problem in details and proposed number of improvements, that made proving optimality of solutions tractable for medium-sized instances.

Keywords: nurse rostering problem, branch and price, combinatorial optimization

Abstrakt

Tato práce popisuje exaktní algoritmus pro problem rozvrhování zdravotních sester založený na obecné dekompoziční metodě Branch and Price. Popsaný přístup zvládá širokou škálu měkkých omezení a je vhodný pro nasazení v reálných prostředích, což je doloženo testováním na veřejných instancí problému z reálných prostředích. Mimo jiné jsme ukázali, jak řešení příbuzných těžkých optimalizačních úloh mohou pomoci pro řešení těchto problémů v budoucnu. Studovány byly také vlastnosti problému, na základě kterých jsme navrhli vylepšení, které vedly k dosažitelnosti prokazování optimality na instancích střední velikosti.

Klíčová slova: problém zdravotních sester, branch and price, kombinatorická optimalizace

Contents

List of Figures

1 Introduction

Assigning employees to shifts in workplaces is a problem that is being solved every day on the entire planet. Usually, every place that is running operations involving larger number of people creates a working schedule for them, typically a few weeks onwards. Such working schedule specifies what the duty of an employee is for any given day. Knowing this in advance is important both for employees to arrange their personal activities and for the manager and the company to plan the production.

Imagine for example a department in a large-sized hospital. The task of the head nurse is to ensure that the department is able to provide high-quality health care every day of the year. It means that employees of desired qualification come to work every day so that there is always e.g. someone who can operate an X-Ray machine and that there are at least 5 nurses in the morning and at least 3 nurses in the afternoon. Furthermore, the head nurse has to ensure that employees are not tired and can perform to their best.

However, the hospital has limited staff only. These people also have custom working contracts specifying different number of hours per month they are allowed to work. Moreover, some of the employees have planned vacation, some nurses cannot work at night or they would miss the last bus home, and some nurses have a small child, so they have to take them to school every morning.

The head nurse cannot plan shifts for employees in an arbitrary way. Hospital staff does not want to work too long without a rest period and, moreover, there are also working regulations given by the working union and the labor code which have to be obeyed.

Although it may seem that the motivation includes the nurses in hospitals only, the converse is true. The same problem, scheduling of human resources, is appearing frequently in production, services and other areas of industries where a large number of employees occur. For example, similar problem is being solved in large modern cinemas. A cinema typically employs tens of employees which need to be scheduled over a time period. There are different shift types (morning, day and night shift) and people are trained for different positions (selling tickets, selling food and drinks, welcoming customers, etc.). Moreover, different staffing demands are imposed for different times and days. This and other similar problems can be easily reduced to our problem. Therefore, they can be solved by the same algorithms. Thus, Nurse Rostering Problem is not an artificial example just made up by researches, but it addresses the very important and practical problem of general personnel rostering.

If the (e.g. monthly) schedule for all employees has to be created by hand, the person in charge of it will not only spend a lot of time while doing that but the solution is likely to be far from optimal. The solution by hand can easily take hours to get while a computer might be able to find a better one in the order of minutes. Thus, it makes sense to pass this task over to the computer.

However, this task is hard even for current computers. Properties of Nurse Rostering Problem make it fundamentally different from problems like sorting a set of numbers (which is still time demanding for a human being, but computers are able to do it quickly) or finding a book in a digital library which can be solved easily. Informally said, Nurse Rostering Problem reaches a level of complexity which our computers are not able to solve fast in general. This complexity level is called $\mathcal{NP}$-hard. Simply said, if a problem is $\mathcal{NP}$-hard, it means that, in general, we do not know how to solve it in polynomial time[1] with respect to input length. Commonly, solution of these hard problems takes a long (exponential) time. Nowadays, only algorithms with exponential time complexity for solving $\mathcal{NP}$-hard problems are known.

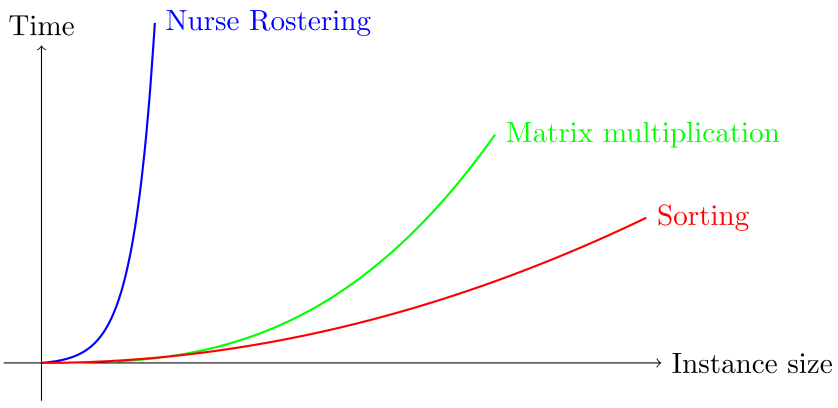


Figure 1.1: Illustrative comparison of computation runtime of polynomial algorithms (matrix multiplication, sorting) and an exponential one (Nurse Rostering).

When facing such a complex problem, the *exponential blow up* must be pushed to the right side (extending the plateau where instances can be solved reasonably fast) as far as possible (see Figure 1.1), so that problems of desired sizes can be solved in reasonable time. It can be done by studying the problem carefully in detail and looking for its properties and structure that can be exploited.

Even though we call this problem Nurse Rostering due to historical reasons (nurses have one of the most complicated labor codes, thus the study of this domain is challenging), one has to keep in mind that many personnel rostering problems are reducible to our problem.

The last question to answer is whether the problem introduced above needs to be solved optimally or if a heuristic (suboptimal) solution is sufficient. Many will argue that *any* solution for a rostering problem is essentially sufficient when one is able to obtain it quickly. However, we claim that if we were able to get the *optimal* solution in a comparable time, there is no

[1]Polynomial algorithm is commonly considered a *fast* one.

reason to use the suboptimal solution.

In this thesis, we study the general variant of Nurse Rostering Problem (Burke et al., 2004) and we show its properties that allow us to solve real-world instances optimally in the order of minutes.

1.1 Problem statement

We are given a planning horizon of length n, a set of employees $\mathcal{E}$, a set of skill competencies $\mathcal{J}$ and a set of a shift types $\mathcal{K}$. For each employee $i \in \mathcal{E}$ we have a set of *patterns* $\mathcal{P}_i$ which defines both desirable and undesirable shift sequences. Each pattern is defined by a regular language and is associated with a start day, a lower and upper limit for its appearance in i's individual schedule and a cost and penalty function for violating it (see Table 1.1). Each employee can also specify their preference for assignment to each shift type on each day. Moreover, employee i can be associated with a permitted *workload.* A workload specifies a minimum and maximum number of time units (thus each shift type $k \in \mathcal{K}$ is associated with some time units, i.e. *shift duration*) allowed and the penalty for violation. Since the workload can be also specified over subintervals of the planning horizon, an employee can be associated with multiple workload constraints simultaneously.

The set of pattern based constraints $\mathcal{P}_i$ and the workload constraints combined together is called a *working contract.* In general, there are two types of constraints — hard constraints and soft constraints. Hard constraints must be satisfied at all cost. These can be some forbidden shift patterns given by a labor code or minimal workload specified by a working contract. Different hard constraints are given by the limitations of real world (i.e. every employee can serve at most one shift at the same time) or by the problem description (i.e. every employee has exactly one skill).

Soft constraints are the ones that can be violated for some penalty. In Nurse Rostering Problem, these are *pattern based* constraints in most cases. Table 1.1 provides an example of pattern based soft constraints and calculation of the penalty. The other common soft constraint is the maximal workload. In contrast to the minimal workload constraint, working overtime is common. However, the employer must pay to those employees more money, which is undesirable from the company's budget point of view.

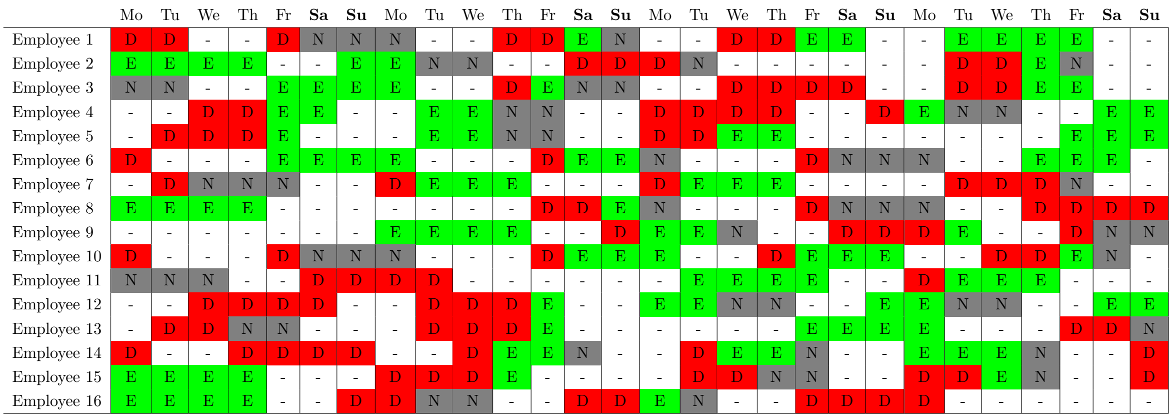
Moreover, we are given a requirement for the preferred coverage R_{mjk} associated with the day m, served by employees with the skill competency j on the shift k for each day of the planning horizon. It is linked with the penalty c^u_{mjk} for understaffing and c^o_{mjk} for overstaffing it. We can also be given a minimum and maximum staffing MI_{mjk} (MA_{mjk} respectively) and penalties t^u_{mjk} and t^o_{mjk} for violation. When both preferred and minimal/maximal coverages

Description	**Pattern**	**Penalty**	**Schedule**	**Cost**
Max 3 consecutive N No 4 N	`NNNN`	5, linear function	$\boxed{N}\ \boxed{N}\ \boxed{N}\ \boxed{N}\ \boxed{N}$	10
No N after E No EN	`EN`	20, linear function	$\boxed{N}\ \boxed{E}\ \boxed{E}\ \boxed{E}\ \boxed{N}$	20
Max 2 days off No - - -	`- - -`	2, quadratic function	$\boxed{N}\ \boxed{-}\ \boxed{-}\ \boxed{-}\ \boxed{-}$	8

Table 1.1: Examples of pattern soft constraints for the individual employee and their calculation. `N` stands for night shift, `E` for early shift and `-` denotes a day off.

are specified, the combined penalty function is used. Notice, that these are soft constraints also. For detail treatment of coverage penalties see Figure 4.1.

The objective is to minimize the sum of constraint violations (patterns) for each employee plus sum of violations of their shift requests plus sum of violations of coverage constraints. Nurse Rostering Problem is commonly modeled by a *nurse-day view* (Cheang et al., 2003). That is matrix **X** where each row corresponds to a nurse and each column to a specific day. Thus, nurse i is assigned to shift x_{ij} on day j. Such matrix of the solved problem can be seen in Figure 1.2.



	Mo	Tu	We	Th	Fr	**Sa**	**Su**	Mo	Tu	We	Th	Fr	**Sa**	**Su**	Mo	Tu	We	Th	Fr	**Sa**	**Su**	Mo	Tu	We	Th	Fr	**Sa**	**Su**
Employee 1	D	D	-	-	D	N	N	N	-	-	D	D	E	N	-	-	D	D	E	E	-	-	E	E	E	E	-	-
Employee 2	E	E	E	E	-	-	E	E	N	N	-	-	D	D	D	N	-	-	-	-	-	-	D	D	E	N	-	-
Employee 3	N	N	-	-	E	E	E	E	-	-	D	E	N	N	-	-	D	D	D	D	-	-	D	D	E	E	-	-
Employee 4	-	-	D	D	E	E	-	-	E	E	N	N	-	-	D	D	D	D	-	-	D	E	N	N	-	-	E	E
Employee 5	-	D	D	D	E	-	-	-	E	E	N	N	-	-	D	D	E	E	-	-	-	-	-	-	-	E	E	E
Employee 6	D	-	-	-	E	E	E	E	-	-	-	D	E	E	N	-	-	-	D	N	N	N	-	-	E	E	E	-
Employee 7	-	D	N	N	N	-	-	D	E	E	E	-	-	-	D	E	E	E	-	-	-	-	D	D	D	N	-	-
Employee 8	E	E	E	E	-	-	-	-	-	-	-	D	D	E	N	-	-	-	D	N	N	N	-	-	D	D	D	D
Employee 9	-	-	-	-	-	-	-	E	E	E	E	-	-	D	E	E	N	-	-	D	D	D	E	-	-	D	N	N
Employee 10	D	-	-	-	D	N	N	N	-	-	-	D	E	E	E	-	-	D	E	E	E	-	-	D	D	E	N	-
Employee 11	N	N	N	-	-	D	D	D	D	-	-	-	-	-	-	E	E	E	E	-	-	D	E	E	E	-	-	-
Employee 12	-	-	D	D	D	D	-	-	D	D	D	E	-	-	E	E	N	N	-	-	E	E	N	N	-	-	E	E
Employee 13	-	D	D	N	N	-	-	-	D	D	D	E	-	-	-	-	-	-	E	E	E	E	-	-	-	D	D	N
Employee 14	D	-	-	D	D	D	D	-	-	D	E	E	N	-	-	D	E	E	N	-	-	E	E	E	N	-	-	D
Employee 15	E	E	E	E	-	-	-	D	D	D	E	-	-	-	-	D	D	N	N	-	-	D	D	E	N	-	-	D
Employee 16	E	E	E	E	-	-	D	D	N	N	-	-	D	D	E	N	-	-	D	D	D	D	-	-	-	-	-	-

Figure 1.2: Example solution of Valouxis instance (Valouxis and Housos, 2000).

We call this matrix the *roster*. When we speak about the *individual schedule* of an employee, we are referring to the corresponding row in the roster.

1.1.1 Difficulties in practice

Many researchers did not address the problem in a way closely related to the real-world environments (Burke et al., 2004). Since the problem is too complex, algorithms tend to adopt simplifying assumptions making the problem easier to solve. A large number of approaches assume a custom (hand-picked) set of constraints or treat them as hard ones. These assumptions often make problems not well-suited for different environments — it is common that hospital's departments have different requirements and staff's habits, therefore, some practice which is forbidden at some places may be allowed in other ones.

However, due to a large number of restrictions for individual schedules it can easily happen that no solution can meet all the demands, i.e. the problem becomes *overconstrained*. In order to find at least some solution, we have to allow to violate constraints for some *penalty*. Thus, it is reasonable instead of (hard) constraints to assume *soft constraints* (the ones that can be violated even though it is not preferred).

Treating constraints as soft ones gives employer sufficient freedom for the design of the constraint set and other requirements given by the problem description. On the other hand, the issue with defining costs for violations arises. The individual nurse's preferences from their point of view can be more important than for a head nurse who is responsible for a proper staffing. The other obvious issue is that violation of some constraints is hardly comparable (e.g. how to compare penalty for understaffing on the morning shift to the violation of working regulations?).

The usage in practice of flexible approaches allowing soft constraints can be difficult due to questions raised above. However, we believe that the high quality of rosters produced by those algorithms greatly pays off for itself (especially when the solutions are optimal ones like with our approach) and makes flexible approaches useful even though some parameters (penalties) must be supplied.

1.1.2 Computational complexity

The task of solving the problem described above even for just one employee can be identified as $\mathcal{NP}$-hard. It can be shown that reduction from 3-SAT to NRP takes polynomial time (see Chapter 3.2). This means that we cannot find a polynomial algorithm for solving NRP unless $\mathcal{P} = \mathcal{NP}$.

Fortunately, we can exploit the inner structure of the problem in order to speed up the algorithm. Complex constraints for the individual schedules are specified for single employee only, thus these constraints can be resolved independently. These individual employees' schedules are then tied up with coverage constraints which are much simpler to solve. This structure results in a block-diagonal constraint matrix in IP (integer programming) formulation, which can be solved much faster than some dense, highly coupled problems.

1.2 Related work

Nurse Rostering Problem has a long history counting more than 40 years. Methods developed so far can be split into two categories — heuristic and exact. Naturally, the heuristic ones were adopted earlier because they are able to handle larger instances in tractable times. Heuristic approaches for Nurse Rostering Problem cover metaheuristics, methods based on mathematical programming and many others.

The other category of approaches, the exact ones, is considerably smaller. Most of the authors did some simplifying assumptions on the constraints or the staff requirement in order to solve the problem. For example, soft constraints are not always assumed. The issue is that most of these modifications simplify the problem too much to be applied successfully in practice, i.e. in real-world environments.

Essentially, all approaches published until recent time are the heuristic ones or assumed a limited constraint set. Those which did not, will be also described in the paragraphs below.

The (Warner and Prawda, 1972) and (Warner, 1976) are early publications on a heuristic method based on the mathematical programming for Nurse Rostering Problem. It allowed to specify weights for constraints (undesired patterns) in the schedule. Some assignments were made by hand (shifts at weekends etc.) before the optimization. The algorithm had 2 stages — it searched for a feasible solution first and then it tried to find an improvement. This algorithm was employed in several hospitals across the U.S.

In Nurse Rostering Problem, there are obviously often many conflicting goals — e.g. to minimize the difference between desired and scheduled staff coverages, to maximize personal preferences, to minimize labor costs, etc. Thus, some multi-criteria approaches emerged. (Jaszkiewicz, 1997) used a semi-interactive approach. The method used two stages — the first stage ran simulated annealing (pareto-simulated annealing) and the second one was an *interactive* stage, where the human operator selected good solutions by hand. Therefore, the scheduling process was not fully automated.

(Jaumard et al., 1998) showed an approach based on column generation and branch and bound. The objective was to minimize salary cost and employee dissatisfaction subject to coverage demands. The subproblem (finding schedule for given nurse) was a *resource constrained shortest path* problem with a non-decreasing resource function (Irnich and Desaulniers, 2005).

Authors assumed a fixed constraint set including workload, shift rotations, etc. Their paper reported that the proposed method was olny able to find feasible solutions, not necessary optimal ones. However, this work was an important improvement towards the exact approaches.

For solving real-world problems, a considerable number of metaheuristic approaches was successfully applied. (Burke et al., 1999) introduced a hybridized tabu search and human-inspired techniques for an improvement of solutions. (Burke et al., 2001) described a set of genetic and memetic algorithms where coverage constraints are satisfied throughout the whole search. The proposed method handled complex constraints and requirements for staffing. The introduced recombination operator was based on hand-crafted characteristics of the solutions. However, the approach suffered from long running times. (Aickelin and Dowsland, 2000, 2004) proposed an indirect genetic algorithm where the constraints are expressed as a set of feasible patterns. (Ikegami and Niwa, 2003) used a mathematical programming formulation solved by tabu search. The approach was used in order to solve problems in Japan's hospitals, thus it implemented specific work regulation requirements. The problem is decomposed into subproblems, where all but one nurse are fixed. However, the proposed approach is also heuristic. (Burke et al., 2010) introduced hybridized heuristic ordering with variable neighborhood search (Mladenović and Hansen, 1997). The method is able to support rostering decisions in large modern hospital environments. The proposed approach offered good balance between quality of solutions and computation time, allowing expressive problem formulations.

(Petrovic et al., 2003) used case-based reasoning, where the problems are solved based on past experience. The approach imitated the human style of reasoning about rostering (Burke et al., 2004). Only a limited set of rules for the roster was permitted, although it was applied in practice.

The (Menana and Demassey, 2009) showed an interesting improvement for the Constraint Programming model, where undesired patterns in the individual schedules are modeled as fragments of a regular language. Authors designed a custom global constraint encapsulating the `regular` and `gcc` (global cardinality constraint) global constraints in order to speed up domain propagation. Moreover, they used Lagrangian relaxation for filtering domains. It greatly improved runtime, however, the model handled hard constraints only.

(Maenhout and Vanhoucke, 2010) described an exact method based on the branch and price algorithm. The proposed model allowed preferred coverage requirements (2-piecewise linear penalty) over single skill competency and assumed hard constraints only, which allowed reasonable solving times. Moreover, authors explored various branching strategies. They used a custom artificial dataset for testing performance.

A branch and cut method was used for solving one of the modifications of Nurse Rostering Problem in International Nurse Rostering Competition 2010 in (Santos et al., 2014). The authors implemented a custom cut generation procedure for *clique cuts*. They proposed an approach where instead of finding the most violated cut they add all violated cuts at once. (Burke and Curtois, 2014) presented a method based on branch and price algorithm. It was able to solve real-world benchmark instances (Curtois, 2014). Their algorithm handled various soft constraints and expressive problem formulations. However, even though authors used the branch and price algorithm, which in general is an exact method, Burke's method is not exact in our opinion. We present arguments supporting this claim in Chapter 1.3.

1.3 Summary

Nurse Rostering Problem is greatly complicated. As (Tien and Kamiyama, 1982) mentioned: *"nurse rostering is more complex than the traveling salesman problem . . . "*. The development of an exact algorithm, which is able to solve real-world instances up to the optimality is a rather challenging task. Thus most of the approaches to Nurse Rostering Problem are heuristic — *"most of these approaches are not exact" . . . "problem is too complex"* (Burke et al., 2004).

The objective is to minimize the sum of the penalties for violation of individual preferences, work regulations, coverage constraints and shift patterns in individual schedules in most cases. Thus, it is desired to treat constraints as soft ones. In our opinion, it is a more faithful description of reality. However, it makes the problem more difficult. Although it may not be obvious, problems with soft constraints are harder to solve than the ones with hard constraints only. When one is solving the problem optimally, hard constraints can be used for effective pruning of the search space, which cannot be done with soft constraints easily.

When one is looking for the best solution of the roster, the aim is to design an algorithm for *proving* its optimality in a short runtime. In order to find the optimal solution, the solution with the lowest objective value has to be found and moreover, the proof that there is no other with lower objective value is needed. This is hard in this domain and it makes the problem challenging. To get some sense about particular numbers, the typical instance has about 20 employees, a 4 week planning horizon with 4 shift types and about 15 soft constraints per employee. This forms a search space of a size approximately $2^{1120} \approx 10^{337}$ where each state (solution) is feasible. In order to solve such instances up to the optimality, one has to reason intelligently about which parts of the search space are actually useful and prune other parts aggressively.

For finding the exact solution of Nurse Rostering Problem, the integer programming is usually a method of choice. In our opinion, the branch and price approach is more flexible than the branch and cut. It permits more complicated constraints, which could be hard to put into general integer programming model. Therefore, the branch and price approach is more suitable for solving real-world instances.

To the best of our knowledge, the only work on Nurse Rostering Problem solved by the branch and price approach which is evaluated on real-world benchmark instances (Curtois, 2014) was published by (Burke and Curtois, 2014). The authors briefly described a basic variant of the branch and price approach but the description of arguably the most important part of the algorithm, the *pricing problem*, lacks details. Moreover, its description contains mistakes — in the *dominance procedure*, lower bound on the reduced cost is not mentioned, which is a crucial detail that was omitted. Moreover, we were not able to reproduce the results achieved by their pricing procedure, thus we assume that some important details were omitted (one clear example is the calculation of the lower bound which is missing completely, see Chapter 3.3.1 for more details). Moreover, authors avoided the statement, whether their approach is exact or heuristic. The article creates the impression that it presents an exact method, since the branch and price is commonly used as the exact one. However, the results show that their algorithm sometimes seemingly randomly stops without reaching any maximal run time limit. The same behavior can be observed in the results for different benchmark data presented in (Curtois T., 2014). In this work, authors more or less admitted that their approach is not able to always find the optimal solution, although time or memory limits are not exceeded. Therefore, it seems that they employed some speedup techniques (e.g. rounding up some decision variables, which prevents from backtracking) while sacrificing the completeness of the algorithm and those techniques were not described in the paper.

Even though their algorithm was able to find the optimal solution for the majority of real-world test instances, it is a simplified task without proving their optimality. To see the difference between *finding* the optimal solution and *proving* its optimality, consider e.g. *Golomb ruler problem* (Smith et al., 1999). For example, the optimal ruler of order of 26 was found in 2007 using a large cluster of computers, but its optimality was proven nearly two years after its discovery[2].

However, (Burke and Curtois, 2014) presents the best known results and we will compare ourselves to those even though it is not clear whether their approach is exact.

1.4 Contribution

We have designed an exact algorithm for solving Nurse Rostering Problem which allows pattern based soft constraints, coverage requirements with 4-piecewise linear penalties, soft workload constraints and employees' preferences. The method is based on the branch and price

[2] `http://blogs.distributed.net/2009/02/24/17/26/bovine/`

approach, whose parts were further improved. We validated results on real-world benchmark data (Curtois, 2014).

Beside achieving better results than in (Burke and Curtois, 2014) we further explored the possibilities of applying the knowledge gained during the run of the algorithm in order to decrease the computation time. Moreover, our contribution consists of solving a new real-world instance for the surgery department of *Motol* hospital in Prague, Czech Republic (see Chapter 7.2). We were able to find the optimal schedule for their department in a few minutes and we improved the overall quality of their roster's objective by a factor of over 220. More specifically, our contribution consists mainly of:

- a new LP formulation of a master model allowing minimal and maximal staffing levels across subsets of skills (see Chapters 4 and 2.4.1) and featuring a novel symmetry breaking technique (see Chapter 6.4)
- a MIP based pricing model with a number of improvements containing branching priorities and dynamical control of precision level based on the convergence of master model (see Chapter 3.3.2)
- the proof that the decision version of the pricing problem with soft constraints is $\mathcal{NP}$-complete and is not polynomially approximable within ϵ (see Chapter 3.2)
- new initiation heuristics (see Chapter 4.2) and new primal heuristics for obtaining upper bounds (see Chapter 4.6)
- a number of improvements to column generation applied to Nurse Rostering Problem, mainly subproblem skipping and earlier branching (see Chapters 6.3.4 and 6.2.3)
- a novel machine learning algorithm for upper bound prediction in the pricing problem (see Chapter 5.2)

1.5 Outline

The rest of the thesis is organized as follows. Chapter 2 presents an overview of the general branch and price algorithm and column generation. Then, we show how to apply it to Nurse Rostering Problem and we derive some important equations. In Chapter 3, the *Pricing Problem* is defined and its computational complexity is proven. We show solution methods, both exact and heuristic ones. It concludes with noteworthy aspects of the problem. Chapter 4 defines the *Master Problem*. Initiation heuristics are discussed as well as various branching schemes. In Chapter 5 we deal with opportunities for application of machine learning methods to problems in combinatorial optimization. We show how the knowledge gained during the column generation may be applied for future problems. Chapter 6 presents topics concerning effective usage of the branch and price algorithm we have discovered. The thesis concludes with experimental evaluation in Chapter 7 and states ideas for future research in Chapter 8.

2 Branch and price method

The purpose of this chapter is to introduce the reader to the column generation and branch and price algorithm as a whole. We will show an interpretation of column generation using duality and the application of branch and price approach to Nurse Rostering Problem. The more detailed treatment of branch and price approach to the general integer programming problem is nicely described in (Barnhart et al., 1998).

2.1 General overview

This chapter describes the branch and price method (Barnhart et al., 1998). It consists of several parts, which will be later described individually in separated chapters. The diagram of the algorithm can be seen in Figure 2.1. Before the problem can be solved by branch and price, one has to decompose the original problem (*original formulation*) to a *master problem.* Informally, master problem is a linear program that consists of variables related to all feasible states of a state space given implicitly by the original formulation. This usually means, that the master problem is huge (typically it has exponentially many variables in terms of number of original formulation's variables).

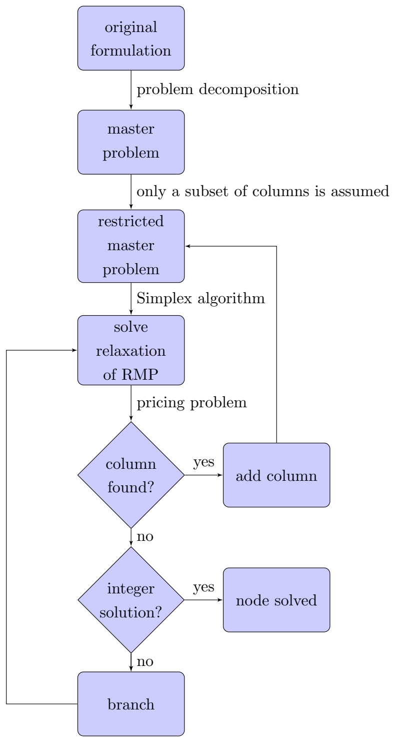


Figure 2.1: Block diagram of the branch and price algorithm.

However, since only a few of these new variables are non-zero in an optimal solution, it is natural to not consider them all at once. When only a subset of them is considered, we get a *restricted master problem*. Throughout the optimization process, we repeatedly insert new variables (*columns*) to the restricted master problem until the point, where one is able to prove, that it represents an optimal solution of the master problem itself (i.e. no additional columns are required).

In case that this solution is integer, it also represents a solution of the original formulation. If it is not, one has to split the search space into disjoint parts and repeat the procedure for each part. By examination of all nodes which yielded with an integer solution we can ensure that we find the optimal one.

2.2 Original formulation

The problem described in Chapter 1.1 can be modeled as an integer programming problem. The most straightforward formulation is to use binary decision variables for assigning people to a shift on specific day. We will refer them as $\mathbf{x}$ in the rest of the text. A model based on these decision variables is not very good for two reasons — first of all, it contains a lot of symmetries (see Chapter 6.4 for more details) and secondly, it has weak linear relaxation.

We will show both the original formulation of the problem and its decomposition on a simple example. Consider following problem — we have 2 identical employees to be scheduled over 4 days. There are only day shift D and day off (denoted as $-$). Each employee has constraints that they can work only 1 or 2 days and can has 2 or 3 days off. Each violation of these constraints is penalized with 1 (unit penalty). For each day we require that there is at least one employee on D shift with penalty 5 for understaffing.

Using the original $\mathbf{x}$ variables, we could model this problem in following way

$$\min_{\mathbf{x},\mathbf{p},\mathbf{n}} \sum_{e\in\{1..2\}} p^o_{e,0} + p^o_{e,1} + p^u_{e,0} + p^u_{e,1} + \sum_{m\in\{1..4\}} 5n^u_{m,1} \tag{2.1}$$

subject to

$$\forall e \in \{1..2\} : \sum_{m\in\{1..4\}} x_{em0} \geq 2 - p^u_{e,0} \tag{2.2}$$

$$\forall e \in \{1..2\} : \sum_{m\in\{1..4\}} x_{em0} \leq 3 + p^o_{e,0} \tag{2.3}$$

$$\forall e \in \{1..2\} : \sum_{m\in\{1..4\}} x_{em1} \geq 1 - p^u_{e,1} \tag{2.4}$$

$$\forall e \in \{1..2\} : \sum_{m\in\{1..4\}} x_{em1} \leq 2 + p^o_{e,1} \tag{2.5}$$

$$\forall m \in \{1..4\} : \sum_{e \in \{1..2\}} x_{em1} \geq 1 - n^u_{m,1} \tag{2.6}$$

$$\forall e \in \{1..2\}, \forall m \in \{1..4\} : x_{em0} + x_{em1} = 1 \tag{2.7}$$

$$\mathbf{x} \in \{0,1\}^{2\times 4\times 2} \tag{2.8}$$

$$\mathbf{p}, \mathbf{n} \geq \mathbf{0} \tag{2.9}$$

where variable x_{emk} is equal to 1 if and only if the employee e is assigned to shift k (denoting $k = 0$ as day off) on day m. Variable $p^u_{e,k}$ ($p^o_{e,k}$) is associated with the penalty for violations of lower limit (upper limit respectively) for the number shift k for employee e. Finally, $n^u_{m,1}$ is slack variable for understaffing on day m.

Constraints (2.2) through (2.5) compute quality of individual schedules via penalties $\mathbf{p}$ and constraint (2.6) is defining the penalty for understaffing on each day.

Instead of above formulation, we will use more efficient one by exploiting special structure in the problem making it solvable efficiently.

2.3 Reformulation for Nurse Rostering Problem

The same problem as in previous chapter can be formulated in following way

$$\min_{\mathbf{y},\mathbf{n}} \sum_{i=1}^{2} \sum_{j=1}^{16} c_{ij} y_{ij} + \sum_{m=1}^{4} \sum_{k=1}^{2} 5 n^u_{mk} \tag{2.10}$$

subject to

$$\forall i \in \{1..2\} : \sum_{j=1}^{16} y_{ij} = 1 \tag{2.11}$$

$$\mathbf{n} \geq \mathbf{0} \tag{2.12}$$

with variables $y_{i,j}$ in Figure 2.2.

c_{ij}:	2	0	0	0	0	0	0	2	0	0	0	2	0	2	2	4
	-	-	-	-	-	-	-	-	D	D	D	D	D	D	D	D
	-	-	-	-	D	D	D	D	-	-	-	-	D	D	D	D
	-	-	D	D	-	-	D	D	-	-	D	D	-	-	D	D
	-	D	-	D	-	D	-	D	-	D	-	D	-	D	-	D
	$y_{1,1}$	$y_{1,2}$	$y_{1,3}$	$y_{1,4}$	$y_{1,5}$	$y_{1,6}$	$y_{1,7}$	$y_{1,8}$	$y_{1,9}$	$y_{1,10}$	$y_{1,11}$	$y_{1,12}$	$y_{1,13}$	$y_{1,14}$	$y_{1,15}$	$y_{1,16}$

c_{ij}:	2	0	0	0	0	0	0	2	0	0	0	2	0	2	2	4
	-	-	-	-	-	-	-	-	D	D	D	D	D	D	D	D
	-	-	-	-	D	D	D	D	-	-	-	-	D	D	D	D
	-	-	D	D	-	-	D	D	-	-	D	D	-	-	D	D
	-	D	-	D	-	D	-	D	-	D	-	D	-	D	-	D
	$y_{2,1}$	$y_{2,2}$	$y_{2,3}$	$y_{2,4}$	$y_{2,5}$	$y_{2,6}$	$y_{2,7}$	$y_{2,8}$	$y_{2,9}$	$y_{2,10}$	$y_{2,11}$	$y_{2,12}$	$y_{2,13}$	$y_{2,14}$	$y_{2,15}$	$y_{2,16}$

Figure 2.2: New variables and their meaning.

Each of these variables is related to some individual schedule for its employee and is associated with the cost of that schedule. Selecting a variable contributes to the coverage constraints included in the model.

$$0y_{1,1} + 0y_{1,2} + \cdots + 1y_{1,16} + 0y_{2,1} + 0y_{2,2} + \cdots + 1y_{2,16} \geq 1 - n^u_{1,1} \tag{2.13}$$

$$0y_{1,1} + 0y_{1,2} + \cdots + 1y_{1,16} + 0y_{2,1} + 0y_{2,2} + \cdots + 1y_{2,16} \geq 1 - n^u_{2,1} \tag{2.14}$$

$$0y_{1,1} + 0y_{1,2} + \cdots + 1y_{1,16} + 0y_{2,1} + 0y_{2,2} + \cdots + 1y_{2,16} \geq 1 - n^u_{3,1} \tag{2.15}$$

$$0y_{1,1} + 1y_{1,2} + \cdots + 1y_{1,16} + 0y_{2,1} + 1y_{2,2} + \cdots + 1y_{2,16} \geq 1 - n^u_{4,1} \tag{2.16}$$

Denoting left-hand-side of constraints (2.13) to (2.16) as matrix $\mathbf{A}$, the entry a_{ij} takes value 1 if the j-th schedule assigns D shift on i-th day and 0 otherwise. One of the optimal solutions of the problem (2.10) is $y_{1,4} = 1$, $y_{2,13} = 1$ with the objective value 0 since it is a lower bound on the objective.

Having all schedules y_{ij} enumerated explicitly wouldn't help us to solve the problem faster since there is an exponential number of them. Instead of it, we will only consider a subset of them and generate new ones on demand. So, ideally, we would generate only exactly 2 of them ($y_{1,4}$ and $y_{2,13}$) out of $2 \times 16 = 32$. Therefore, the linear program would be tiny and it will be solved efficiently.

The example shown above can be generalized as follows (Desrosiers and Lübbecke, 2005). Consider following general form of linear programming problem

$$\min_{\mathbf{x}} \mathbf{c}^T\mathbf{x} \tag{2.17}$$

subject to

$$\mathbf{Ax} \leq \mathbf{b}$$

$$\mathbf{x} \in X = \{\mathbf{x} \in \mathbb{Z}^n \,|\, \mathbf{F}\mathbf{x} \leq \mathbf{w}\}$$

For the decomposition we use the fact that constraints specified by matrix **A** are *easy* to solve on their own but the constraints in matrix **F** are the *difficult* ones. For our problem it holds that **A** $\approx$ coverage constraints and **F** $\approx$ work regulations constraints.

Furthermore, the latter one is a block-diagonal matrix since the work regulations for individual schedules are not coupled to each other. We exploit this structure by following informal description — the *master model* will take care of constraints in the matrix **A** and the so-called *pricing model* will repeatedly solve the task of finding individual schedules for employees such that $\mathbf{F}\boldsymbol{x} \leq \mathbf{w}$ holds. Corresponding *master problem* is then

$$\min_{\boldsymbol{\lambda}} \sum_{l \in \mathcal{L}} c_l \lambda_l \tag{2.18}$$

subject to

$$\sum_{l \in \mathcal{L}} \mathbf{a}_l \lambda_l \leq \mathbf{b} \tag{2.19}$$

$$\forall l \in \mathcal{L} : \lambda_l \geq 0 \tag{2.20}$$

Decomposed model into this structure can be then solved efficiently by *column generation* (Desrosiers and Lübbecke, 2005). This method solves the problem by generating columns into matrix **A** and coefficients into vector **c** iteratively. At each iteration it solves problem based on the dual formulation of (2.18) in form of $\min_{l \in \mathcal{L}}\{c_l - \boldsymbol{\pi}^T \mathbf{a}_l\} = \min_{\mathbf{x} \in X}\{c(\mathbf{x}) - \boldsymbol{\pi}^T \mathbf{a}(\mathbf{x})\}$ subject to that cost c_l of a column $\mathbf{a}_l$ is given by the structure of the problem described by the *difficult* constraints $\mathbf{F}\boldsymbol{x} \leq \mathbf{w}$. The $\boldsymbol{\pi}$ denotes dual prices for the constraints (2.19). Therefore, difficult constraints are not present explicitly in the decomposed model.

In our problem, each column corresponds to a complete working schedule for one employee. Since all the possible columns are not present explicitly in **A**, new ones are added in iterative way. Furthermore, only the schedules, which are useful for the master model according to the coverage constraints can be considered. By adding these useful schedules (columns) iteratively, we converge into the optimal solution without enumerating all of them. This is one of the sources of efficiency of column generation method. More detailed treatment of column generation is presented in Chapter 2.6.

So far, we have talked only about column generation itself. One has to keep in mind, that column generation is a continuous optimization method, thus it does not ensure integer property of the solution. Additional process has to be employed. By splitting search space in a similar way as it is done by the branch and bound method (Lawler and Wood, 1966), we obtain the optimal solution (integer one) of the original problem. Thus, the branch and price method is essentially the branch and bound method where each node (restricted master model) is solved by column generation. The best integer solution found so far serves as the upper bound while the lower bound at each node of a search tree is the objective value of the master problem solved up to the optimality by column generation (see Chapter 4).

The next section describes specific formulation of the master problem for Nurse Rostering Problem. It is a model assigning schedules to employees and calculating coverage constraint penalties for selected schedules. Then, we will show how to find new columns which help us to solve the problem up to the optimality.

2.4 Primal model

As we mentioned in Chapter 1.1, the instance of Nurse Rostering Problem consists (among others) of a set of employees $\mathcal{E}$, their skills $\mathcal{J}$, a set of shifts $\mathcal{M}$, time horizon and preferred/minimal/maximal coverages and their penalties for understaffing/overstaffing for each day, skill and shift combination. These requirements are taken in account by primal (master) model (see Figure 2.3)

$$\begin{aligned}\min_{\mathbf{y},\mathbf{n},\mathbf{v}} \quad \sum_{i\in\mathcal{E}}\sum_{l\in\mathcal{F}_i} c_{il}y_{il} \; &+ \sum_{m\in\mathcal{M}}\sum_{j\in\mathcal{J}}\sum_{k\in\mathcal{K}} c^u_{mjk}n^u_{mjk} + \sum_{m\in\mathcal{M}}\sum_{j\in\mathcal{J}}\sum_{k\in\mathcal{K}} c^o_{mjk}n^o_{mjk} + \\ &+ \sum_{m\in\mathcal{M}}\sum_{j\in\mathcal{J}}\sum_{k\in\mathcal{K}} t^u_{mjk}v^u_{mjk} + \sum_{m\in\mathcal{M}}\sum_{j\in\mathcal{J}}\sum_{k\in\mathcal{K}} t^o_{mjk}v^o_{mjk} \qquad (2.21)\end{aligned}$$

subject to

$$\begin{aligned}
\forall i\in\mathcal{E}: &\quad \sum_{l\in\mathcal{F}_i} y_{il} &&= 1 && (2.22)\\
\forall m\in\mathcal{M},\forall j\in\mathcal{J},\forall k\in\mathcal{K}: &\quad \sum_{i\in\mathcal{E}}\sum_{l\in\mathcal{F}_i} a_{ilmjk}y_{il} + n^u_{mjk} - n^o_{mjk} &&= R_{mjk} && (2.23)\\
\forall m\in\mathcal{M},\forall j\in\mathcal{J},\forall k\in\mathcal{K}: &\quad n^u_{mjk} - n^o_{mjk} - v^u_{mjk} &&\leq R_{mjk} - MI_{mjk} && (2.24)\\
\forall m\in\mathcal{M},\forall j\in\mathcal{J},\forall k\in\mathcal{K}: &\quad -n^u_{mjk} + n^o_{mjk} - v^o_{mjk} &&\leq MA_{mjk} - R_{mjk} && (2.25)\\
\forall i\in\mathcal{E},\forall l\in\mathcal{F}_i: &\quad y_{il} &&\geq 0 && (2.26)\\
\forall m\in\mathcal{M},\forall j\in\mathcal{J},\forall k\in\mathcal{K}: &\quad n^u_{mjk} &&\geq 0 && (2.27)\\
\forall m\in\mathcal{M},\forall j\in\mathcal{J},\forall k\in\mathcal{K}: &\quad n^o_{mjk} &&\geq 0 && (2.28)\\
\forall m\in\mathcal{M},\forall j\in\mathcal{J},\forall k\in\mathcal{K}: &\quad v^u_{mjk} &&\geq 0 && (2.29)\\
\forall m\in\mathcal{M},\forall j\in\mathcal{J},\forall k\in\mathcal{K}: &\quad v^o_{mjk} &&\geq 0 && (2.30)
\end{aligned}$$

Figure 2.3: Primal master model.

where c_{il} is a cost of pattern l for employee i consisting of soft cost penalties (the exact equation for c_{il} is (2.45)) and y_{il} is a variable assigning a schedule l to employee i. The $\mathcal{F}_i$ denotes the set of patterns of employee i. The c^u_{mjk} and c^o_{mjk} are penalties for understaffing (overstaffing respectively) with respect to preferred coverages R_{mjk} at shift k on day m for skill j and the t^u_{mjk} (t^o_{mjk}) is penalty for violation of a minimal coverage MI_{mjk} (maximal coverage MA_{mjk}) at shift k on day m for skill j.

The n^u_{mjk} and n^o_{mjk} are slack variables for understaffing (overstaffing respectively) on day m at shift k for skill j. Similarly, v^u_{mjk} and v^o_{mjk} are also slack variables associated with the staffing levels under (over) minimal (maximal) requirements.

Most importantly, a_{ilmjk} is equal to 1 when employee i in its schedule l is assigned to the shift k on the day m with skill competency j, otherwise it is 0.

The objective is the sum of costs of assigned patterns and sum of penalties for violating coverage constraints. Constraints (2.22) model the requirement that each employee is assigned to the exactly one shift pattern. Constraints (2.23) model preferred coverage requirements for each day, skill and shift. Finally, constraints (2.24) and (2.25) model minimal and maximal coverages and their respective penalties.

2.4.1 Additional coverage constraints

Although the model described in Figure 2.3 is expressive enough to solve most of instances, it is still not the most general one. Some instances pose additional constraints on desired coverages across all the skills. This can be modeled by adding additional constraints like

$$\begin{aligned}
\forall m \in \mathcal{M}, k \in \mathcal{K}: \quad & \sum_{j \in \mathcal{J}} (-n^u_{mjk} + n^o_{mjk}) + w^u_{mk} && \geq && MI_{mk} - \sum_{j \in \mathcal{J}} R_{mjk} && (2.31)\\
\forall m \in \mathcal{M}, k \in \mathcal{K}: \quad & \sum_{j \in \mathcal{J}} (-n^u_{mjk} + n^o_{mjk}) - w^o_{mk} && \leq && MA_{mk} - \sum_{j \in \mathcal{J}} R_{mjk} && (2.32)\\
\forall m \in \mathcal{M}, k \in \mathcal{K}: \quad & w^u_{mk}, w^o_{mk} && \geq && 0 && (2.33)
\end{aligned}$$

and by putting penalties associated with slack variables w^u_{mk} and w^o_{mk} into the objective function. Moreover, we can model other coverage constraints which consider the specific subset of skills $\mathcal{J}$ with similar approach.

These additional constraints has no impact on the form of objective function of the pricing problem — the information about altered coverage requirements is communicated to the pricing subproblem through $\boldsymbol{\pi}$ dual prices.

2.5 Dual model

Using Lagrangean relaxation (Boyd and Vandenberghe, 2004) we can derive following dual formulation of problem shown in Figure 2.3. In Figure 2.4 we can see full form of dual

$$\begin{aligned}\max_{\boldsymbol{\gamma},\boldsymbol{\pi},\boldsymbol{\phi},\boldsymbol{\psi}} \sum_{i\in\mathcal{E}}\gamma_i + \sum_{m\in\mathcal{M}}\sum_{j\in\mathcal{J}}\sum_{k\in\mathcal{K}} R_{mjk}\pi_{mjk} \quad &+ \sum_{m\in\mathcal{M}}\sum_{j\in\mathcal{J}}\sum_{k\in\mathcal{K}}(R_{mjk}-MI_{mjk})\phi_{mjk} + \\ &+ \sum_{m\in\mathcal{M}}\sum_{j\in\mathcal{J}}\sum_{k\in\mathcal{K}}(MA_{mjk}-R_{mjk})\psi_{mjk}\end{aligned}$$

subject to

$$\forall i \in \mathcal{E}: \quad \gamma_i \in \mathbb{R} \tag{2.34}$$

$$\forall m \in \mathcal{M}, \forall j \in \mathcal{J}, \forall k \in \mathcal{K}: \quad \pi_{mjk} \in \mathbb{R} \tag{2.35}$$

$$\forall m \in \mathcal{M}, \forall j \in \mathcal{J}, \forall k \in \mathcal{K}: \quad \phi_{mjk} \leq 0 \tag{2.36}$$

$$\forall m \in \mathcal{M}, \forall j \in \mathcal{J}, \forall k \in \mathcal{K}: \quad \psi_{mjk} \leq 0 \tag{2.37}$$

$$\forall i \in \mathcal{E}, \forall l \in \mathcal{F}_i: \quad \gamma_i + \sum_{m\in\mathcal{M}}\sum_{j\in\mathcal{J}}\sum_{k\in\mathcal{K}} a_{ilmjk}\pi_{mjk} \leq c_{il} \tag{2.38}$$

$$\forall m \in \mathcal{M}, \forall j \in \mathcal{J}, \forall k \in \mathcal{K}: \quad \pi_{mjk} + \phi_{mjk} - \psi_{mjk} \leq c^u_{mjk} \tag{2.39}$$

$$\forall m \in \mathcal{M}, \forall j \in \mathcal{J}, \forall k \in \mathcal{K}: \quad -\pi_{mjk} - \phi_{mjk} + \psi_{mjk} \leq c^o_{mjk} \tag{2.40}$$

$$\forall m \in \mathcal{M}, \forall j \in \mathcal{J}, \forall k \in \mathcal{K}: \quad -\phi_{mjk} \leq p^u_{mjk} \tag{2.41}$$

$$\forall m \in \mathcal{M}, \forall j \in \mathcal{J}, \forall k \in \mathcal{K}: \quad -\psi_{mjk} \leq p^o_{mjk} \tag{2.42}$$

Figure 2.4: Dual master model.

problem. We will use dual formulation in Chapter 2.6.1 for derivation of column generation procedure for our problem.

The most important variables in dual model are π_{mjk}. Each of these variables is associated with the constraint prescribing staffing levels at the related day, shift and skill. When a restricted master problem (the one with fixed number of column) is solved up to the optimality (e.g. by Simplex algorithm), value of π_{mjk} corresponds to algorithm's need to schedule (or even not to schedule) some employees on the corresponding shift/day/skill.

2.6 Column generation

Column generation is a famous technique for solving LPs with large number of variables. The core of the method is not to consider all the variable configurations at once, but rather start with small number of them and generate the new ones on the fly as it is needed in order to improve objective value. The point is that one can prove that all the variables presented in a model are not needed for finding and proving optimal solution but specific subset of them is just enough (Desrosiers and Lübbecke, 2005).

The obvious question could arise, how one can possibly know which new variable should be generated at the given step? The answer lies in the dual formulation of the given problem — find the constraint in the dual problem which corresponds to variable we wish to generate in master problem and find cutting plane in that form such that it will make current dual solution infeasible. This gives us a new variable for the primal program (see Figure 2.5). Procedure repeats until we are able to find some cuts in the dual problem which violate current dual solution. If there is no such a cut, primal (and also the dual) programs are solved up to the optimality.

Now the question is how to find such cuts. In order to do that, one has to design dedicated algorithm taking into account the form of the dual constraint which we want to be violated. We call this task a subproblem (pricing problem) and the corresponding algorithm a subproblem (pricing) solver. Computational complexity of the subproblem solver is problem-specific. For some problems, we can find polynomial time computable subproblems or pseudopolynomial algorithms (for example in *cutting stock problem* where the subproblem is a *knapsack problem* (Korte et al., 2002)). In other cases the complexity can be $\mathcal{NP}$-hard without any known pseudopolynomial algorithm.

$$\left[\begin{array}{cccc|c} c_1 & c_2 & \cdots & c_n & \\ \hline a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{array}\right] \Bigg\} \pi \text{ dual variables} \xrightarrow{\text{pricing solver}} \left[\begin{array}{c} c_{n+1} \\ \hline a_{1,n+1} \\ a_{2,n+1} \\ \vdots \\ a_{n,n+1} \end{array}\right] \xrightarrow{\text{add new column}} \left[\begin{array}{ccccc|c} c_1 & c_2 & \cdots & c_n & c_{n+1} & \\ \hline a_{11} & a_{12} & \cdots & a_{1n} & a_{1,n+1} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & a_{2,n+1} & b_2 \\ \vdots & & \ddots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & a_{n,n+1} & b_n \end{array}\right]$$

Figure 2.5: Column generation procedure. It generates a new column that could improve primal objective based on the current dual solution.

Our subproblem is usually viewed as a *resource constrained shortest path on acyclic graphs problem*. This is identified (Irnich and Desaulniers, 2005, pp. 4) to be in $\mathcal{NP}$-hard complexity class, which is the result consistent with the previous statement that Nurse Rostering Problem is contained within $\mathcal{NP}$-hard complexity class.

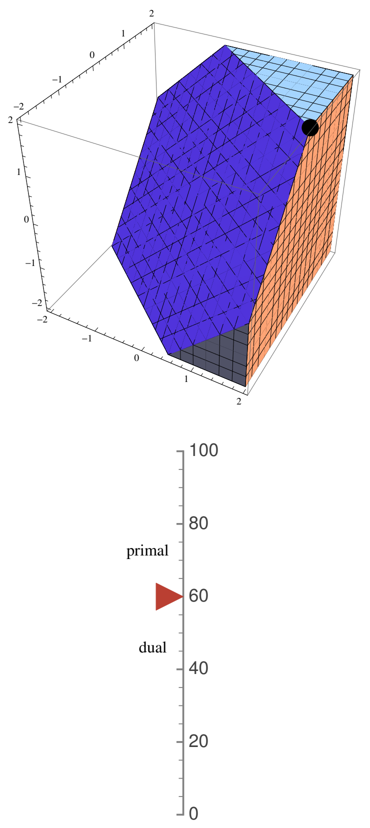
From the LP solver point of view, the column generation procedure is also computationally efficient since we can use previous basis as a starting point for the next iteration with additional variables. Therefore, it is not solving the problem every time from scratch.

One has to keep in mind that column generation does not ensure the integer property of the solution. As we said before, it is the solution method for continuous optimization problems. In our problem we have to ensure integer property by different mechanism called *branching* (see Chapter 4.5).

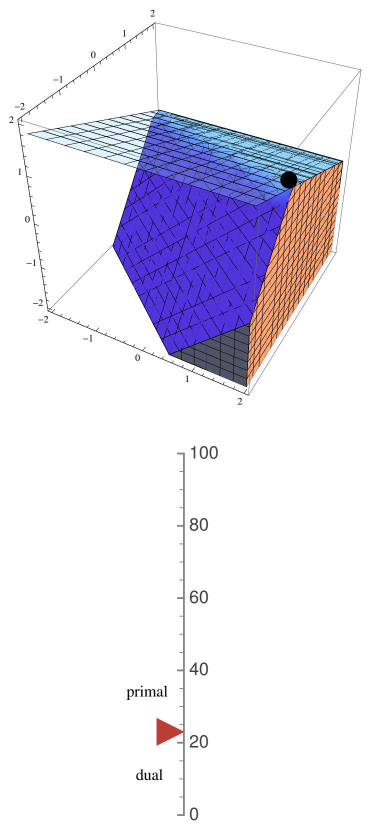
We offer two different perspectives on the intuition why the column generation works in the following paragraphs below. Then, in the next chapter, we formally derive the column generation procedure for our problem.

Primal perspective Column generation generates variables (columns in Simplex tableau) for the primal program. This step cannot increase the objective value since we can always set the new variable y_{il} to zero in order to obtain previous solution which is also feasible in the modified problem. Moreover, selecting (even part of) this variable can decrease the objective value because it could fulfill the coverage constraints (2.23) better.

Dual perspective In the dual task (2.34) we are adding (the most) violated constraints in current dual solution into the model. To see why it is needed, one needs to consider the *weak duality theorem* (Boyd and Vandenberghe, 2004) which states that the objective value of the dual will be always less or equal to the primal objective value (i.e. the dual is a lower bound on the primal). Since we are restricting polyhedron over which the maximization is performed, the dual objective remains equal or decreases. This will weaken a lower bound on the primal objective which consequently can be decreased by the solver. Essentially, we are trying to cut out the optimal dual feasibility polyhedron which corresponds to the optimal set of variables in primal. See Figures 2.6a and 2.6b for the visualization.



(a) Current dual solution



(b) Making dual solution infeasible

Figure 2.6: Optimization in dual space over 3 variables. Displayed polytope is a feasible region of dual LP. Inserting a cutting plane (a new schedule) (see Figure 2.6b) makes current dual solution infeasible, thus it lowers the dual objective value and consequently reduces the primal objective.

2.6.1 Derivation

In order to make current dual solution infeasible, one has to find a cut, which will be violated. We have constraints in form of (2.38) in the dual formulation of the master problem. There is an exponential number of them, however, just a subset of them is active in the optimum. One has to solve a problem of finding $\mathbf{a}_{il}$ in order to find cuts from this set which are violated

$$\gamma_i + \sum_{m \in \mathcal{M}} \sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{K}} a_{ilmjk} \pi_{mjk} > c_{il} \tag{2.43}$$

Obviously, it is equivalent to solving

$$0 > c_{il} - \gamma_i - \sum_{m \in \mathcal{M}} \sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{K}} a_{ilmjk} \pi_{mjk} = \mu_i \tag{2.44}$$

The expression on right-hand-side of (2.44) is called the *reduced cost*, thus we say that we are looking for columns with negative reduced cost ($\mu_i < 0$). The c_{il} is the cost of pattern l under a working contract and shift preferences of employee i. It can be rewritten in terms of original variables as

$$c_{il}(\mathbf{x}) = \sum_{p \in \mathcal{P}_i} c_p^{lb} \max\{0, lb_p - \#p\} + c_p^{ub} \max\{0, \#p - ub_p\} + \sum_{m \in \mathcal{M}} \sum_{k \in \mathcal{K}} p_{imk} x_{mk} \tag{2.45}$$

where p_{imk} is a penalty for not assigning shift k on day m to employee i, $\mathcal{P}_i$ is a set of patterns for employee i, $\#p$ is number of matches of pattern p in schedule $\mathbf{x}$ and c_p^{lb} (c_p^{ub}) is a penalty for violation $\#p$ of bound lb_p (ub_p respectively).

2.7 Algorithmic description

For complete description of the algorithm see pseudocode Algorithm 1.

Algorithm 1: Branch and price

```
1  $initColumns, UB \leftarrow$ heuristics()
2  $queue \leftarrow RMP(initColumns)$
3  while queue is not empty do
4      $RMP \leftarrow queue$
5      do
6          $solution \leftarrow$ solve($RMP$)
7          $y_{il} \leftarrow$ pricing($RMP$)
8          if lb(RMP) $> UB$ then
9              go to 3.
10         end
11         add column $y_{il}$ into the $RMP$
12     while column with negative reduced cost exists;
13     if solution is integer-valued then
14         if $obj < UB$ then
15             $UB \leftarrow obj$
16         end
17     else
18         if $obj \leq UB$ then
19             $queue \leftarrow$ branch($RMP$)
20         end
21     end
22 end
```

Individual parts of the branch and price algorithm are described in the following pages: **heuristics** is described in Chapter 4.2 , **solve** in Chapter 4, **pricing** in Chapter 3, **lb** in Chapter 4.4 and **branch** in Chapter 4.5.

3 Pricing problem

Pricing problem provides new columns which are candidates for entering the Simplex basis and, thus, decreasing the primal objective value. In our case, the pricing problem represents $\mathcal{NP}$-hard problem which has to be solved repeatedly. Since it is used frequently in the branch and price algorithm, its runtime must be as low as possible. In following sections we show different methods for solving it and ways for speeding it up.

This chapter is organized as follows: first, in Chapter 3.1 we formally define the pricing problem, then in Chapter 3.3 we will show different solution methods we have developed, in Chapter 3.4 we will show how the exact pricing method can be turned into an effective heuristics and in Chapters 3.5 through 3.7 we conclude with some theoretical properties of the pricing problem.

3.1 Definition

The pricing problem is given by the following mathematical programming model

$$\min_{\mathbf{x}} -\gamma_i - \sum_{m \in \mathcal{M}} \sum_{k \in \mathcal{K}} \pi_{mk} x_{mk} + c_{il}(\mathbf{x}) \tag{3.1}$$

subject to

$$\forall m \in \mathcal{M} : \sum_{k \in \mathcal{K}} x_{mk} = 1 \tag{3.2}$$

$$\mathbf{x} \in \{0,1\}^{|\mathcal{M}| \times |\mathcal{K}|} \tag{3.3}$$

where $\mathbf{x}$ are *original variables* (assigning days to the shifts), γ_i and π_{mk} are constants in $\mathbb{R}$ and c_{il} is defined as in (2.45).

The description of the equation (2.45) mentions term *number of matching* frequently. One way how to understand it is to treat the employee's schedule as a finite *word* from the alphabet $\mathcal{K}$. A *matching* refers to whether it is contained in regular language defined by a regular expression representing the pattern (see Figure 3.1). Sometimes the pattern is matched multiple times in the schedule, so we sum up occurrences.

$$\cdots \mid - \mid \text{E} \mid \underbrace{\text{E} \mid - \mid \text{N}}_{\text{pattern "\textit{(E or D), Off, N}" is matched}} \mid \text{D} \mid \cdots$$

Figure 3.1: An example of a pattern matching in a schedule.

Due to historical reasons the pricing problem in Nurse Rostering — searching for a schedule for individual employee — is often described as a *resource constrained shortest path* problem (Irnich and Desaulniers, 2005) (see Figure 3.2). The reason is that in the past, most of the approaches did not consider patterns as soft constraints (Burke et al., 2004), therefore, a certain number of appearances of such pattern in a schedule was consider as resource on the path, which was limited (hence the word *constrained*).

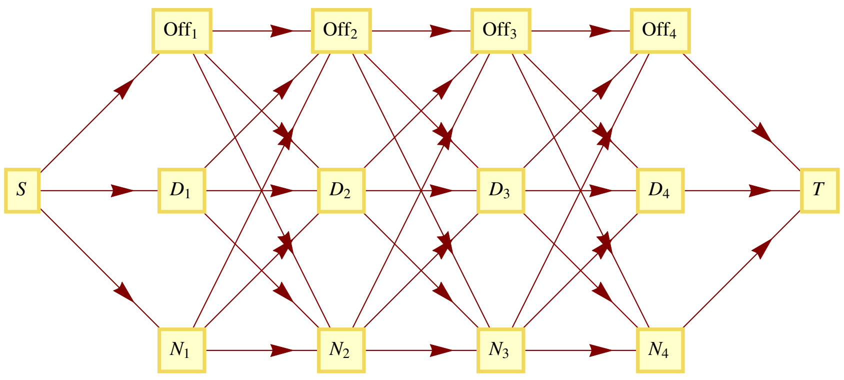


Figure 3.2: Graph structure for the pricing problem. Vertical layers correspond to days and vertices to shift types. We are interested in finding the cheapest $s - t$ path which satisfies given constraints.

However, the modern treatment of Nurse Rostering Problem (see Chapter 1.1) has only a little connection with previous point of view. We consider number of pattern matching in the schedule as a soft constraint, thus we cannot cut off this "path" in a simple way and therefore, it makes only a little sense to imagine them as paths.

In fact, soft constraints are not constraints in a sense that they would restrict the set feasible solutions. They are actually changing the objective function of the problem. If a soft constraint is "violated", it adds a penalty to the objective. Therefore, every soft constraint is actually presented in the objective in some way. Essentially, there are no constraints in our pricing problem, just a complex, non-convex objective function spanning across a space of schedules, which is given by $\mathcal{K}^{|\mathcal{M}|}$. Moreover, the objective function changes during iterations of column generation. Since we consider very general form of patterns defining quality of a schedule, there is no obvious way how to prune the search space.

These properties of the problem hints to the *branch and bound* based solution method. Fortunately, we were able to derive a strong lower bound based on the linear programming relaxation (see Chapter 3.5) which helps significantly to speed up the algorithm. It makes it very fast in practice and furthermore it allows to control the quality of solutions produced by the pricing algorithm (see Chapter 3.4.1).

Moreover, since problems that are being solved by the pricing algorithm are not the *random* ones but rather tend to have some reasonable structure (it is unlikely that individual schedules in optimal roster will contain shift assignments scattered randomly across the planning horizon) we can derive some properties of the objective function *automatically* by machine learning algorithms and apply gained knowledge throughout the solution process (see Chapter 5.2).

3.2 Computational complexity

Most of papers argue about the complexity of Nurse Rostering Problem by stating that for a single employee it is a resource constrained shortest path (RCSP) problem whose complexity is $\mathcal{NP}$-hard as shown in (Irnich and Desaulniers, 2005). However, we were not able to find formal proof of the reduction of the pricing problem with soft constraints to the RCSP. Thus, we will show what is the complexity of decision version of our pricing problem below.

Proposition Determining whether the column with negative reduced cost exists is $\mathcal{NP}$-complete.

Proof We show that 3-SAT[3] is polynomially reducible to the *Pricing Problem*. For each instance of 3-SAT we show how to define an instance of pricing problem such that given 3-SAT instance is satisfiable if and only if the given instance of pricing problem (i.e. column) has a negative reduced cost.

Suppose we have a 3-SAT instance with n literals and m clauses. We define a pricing instance with length of planning horizon n with 2 shift types (e.g. $-$ and D). We set

$$\begin{aligned} \forall m \in \{1 \dots n\}, \forall k \in \{-, D\} : \pi_{mk} &:= 0 \\ \gamma_i &:= -0.5 \end{aligned}$$

For each clause p we define a *match* with lower bound $lb_p := 1$ and upper bound $ub_p := 3$ with costs $c_p^{lb} = c_p^{ub} := 1$. Each match has exactly 3 patterns to be matched — for each literal x_i in clause p we add a pattern D with starting day i and for literals in negative sense $\neg x_j$ a pattern $-$ with starting day j. This reduction is clearly polynomial.

Now we show the problem equivalence. Suppose we have a satisfiable formula in 3-CNF. Therefore, at least one literal in each clause p must be true. Since it holds, corresponding pattern will be matched, thus the lower bound of the match lb_p will not be violated. When we consider the definition of the soft cost penalties of the pricing problem (2.45), it is easy

[3]It is the problem to decide whether the given propositional logical formula in 3-CNF is satisfiable.

to see that there exists a column with reduced cost -0.5, i.e. it is the negative one. In other way around, suppose that a column with negative cost exists. Since this will only happen when there are no soft constraints violated (at least one pattern is matched in each match), it implies that every clause of the 3-CNF formula is true, i.e. the formula is satisfiable.

Up to this we showed that 3-SAT $\rhd_p$ Pricing Problem. Now we show that Pricing Problem is contained in $\mathcal{NP}$ which concludes the proof. This is easy to see, since we have p matches where each of them can be evaluated in constant time. Since p is polynomial in n we compute reduced cost in polynomial time and compare it with 0. ■

Proposition Pricing Problem is not polynomially approximable within $\epsilon > 0$ unless $\mathcal{P} = \mathcal{NP}$.

Proof Using reduction of 3-SAT problem to the Pricing Problem shown above we will show that MAX 3-SAT[4] Problem polynomially reduces directly to the Pricing Problem. We construct an instance of the Pricing Problem such that number of unsatisfied lower bounds on pattern matches is minimized. Since MAX 3-SAT is not polynomially approximable within ϵ (Pardalos and Xue, 1994) we conclude that Pricing Problem cannot be polynomially arbitrarily approximated also. ■

3.3 Exact solution methods

Having an exact solution method for pricing problem is necessary — at some stage algorithm has to prove that restricted master problem converged, i.e. no column with negative reduced cost exists (see diagram in Figure 2.1). As we proved above, this task is $\mathcal{NP}$-complete, thus it is hard to solve it in reasonable time like we demand.

3.3.1 Branch and bound with domination rules

We have designed an exact algorithm based on branch and bound method with dominance rules on partial solutions. It prunes nodes in the search tree (i.e. partial solutions) not only based on lower and upper bounds but also by reasoning about their structure.

For each partial solution it keeps track on the number of matching of each pattern and lower bound on its objective value. Based on the structure of patterns, one is able to reason

[4]It is the problem where we want to maximize the number of satisfied clauses in given 3-CNF propositional logical formula.

whenever for given two partial solutions of the same length one dominates other (i.e. its objective cannot be worse than the other) or if they are incomparable. When one of the partial solutions is dominated by another one, we can prune whole subtree of that solution (see Figure 3.3). The efficiency of the algorithm depends on the power of the domination technique.

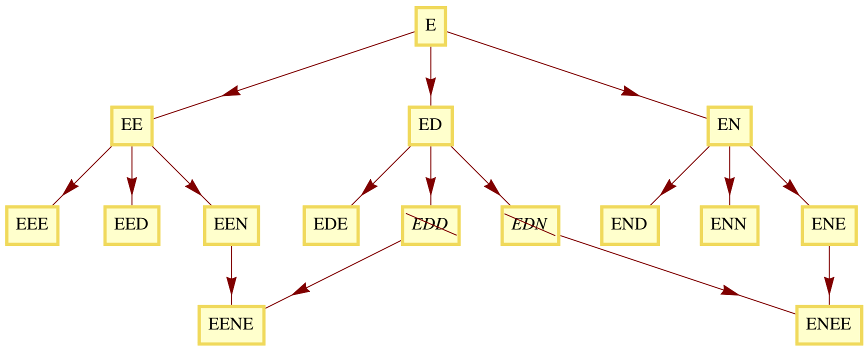


Figure 3.3: An illustrative example of the search tree with some dominated partial solutions. Consider a constraint where we want to have at least two `E` shifts in schedule. Partial schedule `EDD` is dominated by `EEN`, `EDN` is dominated by `ENE`.

However, since we are mostly dealing with soft constraints, design of such procedure is not trivial. We were able to develop following method.

Dominance rule Our dominance procedure compares two partial solutions of the same length. The domination is resolved based on the number of matches of each pattern. We divide patterns into two groups and patterns from each group are treated from two different perspectives — constraint on minimal and constraint on maximal matching. We frequently use the fact that number of pattern matches is non-decreasing. This is easy to see since adding a new shift to the end of the partial schedule cannot undo the pattern match done before, therefore it can only stay the same or increase.

N	E	–	N	···
π_{1-}: 0	π_{3-}: 0	π_{3-}: 0	π_{4-}: 0	···
π_{1E}: +100	π_{2E}: -100	π_{3E}: +100	π_{4E}: +100	···
π_{1D}: +100	π_{2D}: -100	π_{3D}: -100	π_{4D}: +100	···
π_{1N}: -100	π_{2N}: +100	π_{3N}: +100	π_{4N}: 0	···

Constraints:

a) Min 1 D, Max 3 D

b) Max 1 NE

Figure 3.4: An illustrative example of resources. Partial solution `NE-N` is associated with resource vector $[-200, 0, 1]$. The first element is associated with the partially evaluated reduced cost, the other are resources associated with constraints a) and b). Dominance procedure compares vectors of corresponding partial solutions element by element.

Each partial solution is characterized by a number of matching for each pattern from the set $\mathcal{P}_i$ by a lower bound on its reduced cost. We refer to each of those numbers as a *resource*. A resource in partial solution a is denoted $\#_a$ (see example in Figure 3.4).

As we said, patterns are divided into two groups. The first group contains patterns with unit length and the second one contains patterns spanning across more days. We define dominance on separate resources as follows:

1. lower value of the lower bound on the reduced cost dominates the higher one
2. for patterns of unit length
   (a) minimum matching:
      i. if $\#_a < lb$ and $\#_b < lb$ then the solution with higher $\#$ dominates the other
      ii. if $\#_a < lb$ and $\#_b \geq lb$ then b dominates a
      iii. if $\#_a \geq lb$ and $\#_b \geq lb$ then *equivalent*
   (b) maximum matching:
      i. if $\#_a \leq ub$ and $\#_b \leq ub$ then the solution with lower $\#$ dominates the other
      ii. if $\#_a \leq ub$ and $\#_b > ub$ then a dominates b
      iii. if $\#_a > ub$ and $\#_b > ub$ then the solution with lower $\#$ dominates the other
3. for longer patterns

   Even though patterns of unit length are a special case of longer ones, we treat them separately due to following notion of *future matching*. We can reason about patterns with length greater than 1 based on the possibility they will match on current ending of the partial solution. Suppose a pattern of length k and denote $\forall i \in \{1 \dots k-1\}$ possibilities of overlapping the pattern with the ending of partial solution a as $\#M_a^{(i)}$. Then

   (a) minimum matching:
      i. if $\#_a \geq lb$ and $\#_b \geq lb$ then *equivalent*
      ii. else if $\#_a < lb$ and $\#_b \geq lb$ then b dominates a
      iii. else
         A. if $\exists i, j : \#M_a^{(i)} > \#M_b^{(i)}$ and $\#M_a^{(j)} < \#M_b^{(j)}$ then a and b are *incomparable*
         B. if $\forall i : \#M_a^{(i)} \geq \#M_b^{(i)}$ and $\exists j : \#M_a^{(j)} > \#M_b^{(j)}$ then a dominates b
         C. else *equivalent*
   (b) maximum matching:
      i. if $\exists i, j : \#M_a^{(i)} < \#M_b^{(i)}$ and $\#M_a^{(j)} > \#M_b^{(j)}$ then a and b are *incomparable*

   ii. if $\forall i : \#M_a^{(i)} \leq \#M_b^{(i)}$ and $\exists j : \#M_a^{(j)} < \#M_b^{(j)}$ then a dominates b
   iii. else *equivalent*

4. for periodical patterns

   We denote a number of possible matching of the pattern (based on its starting day and structure) in the rest (non-allocated) of the schedule a as $\#F_a$.

   (a) minimum matching:
      i. if $\#_a < lb$ and $\#_b < lb$ then the solution with higher $\#$ dominates the other
      ii. if $\#_a < lb$ and $\#_b \geq lb$ then b dominates a
      iii. if $\#_a \geq lb$ and $\#_b \geq lb$ then *equivalent*

   (b) maximum matching:
      i. if $ub \geq \#_a + \#F_a$ and $ub \geq \#_b + \#F_b$ then
         A. if $\#_a \leq ub$ and $\#_b > ub$ then a dominates b
         B. if $\#_a > ub$ and $\#_b \leq ub$ then b dominates a
      ii. if $\#_a > ub$ and $\#_b > ub$ then the solution with lower $\#$ dominates the other
      iii. else a and b are *incomparable*

When the solutions are *incomparable* then they cannot dominate each other. However, when the procedure states *equivalent* it means, that both resource values do not contribute to the final resolution of the dominance procedure, therefore the particular resources can be omitted.

Finally, partial solution A *dominates* partial solution B if and only if A is not dominated on any resource by solution B and A dominates B on at least one resource. One can show, that dominance procedure defined as above always preserves at least one optimal solution.

Search procedure The search procedure of the algorithm works as described in (Burke and Curtois, 2014). Essentially, it is a breadth-first search with limited number of expanded solutions in a queue. The queue contains a *Pareto optimal* set of partial solutions of the same length. When the limit of queue size is exceeded at some depth, algorithm stores Pareto optimal partial solutions found so far and continues searching to the last depth level. Then, it updates upper bound with the best complete solution found so far, increases maximal queue size and starts again from the point where it overflowed last time. If no overflow happened, algorithm found at least one optimal solution at the last level.

The weakness of this dominance rule is that it is based on the notion of Pareto optimality (Censor, 1977). With the increasing number of resources (patterns) it is becoming more likely that solutions will be evaluated as incomparable since it is sufficient that a single resource in one solution dominates the resource in the other solution and for different resource vice versa. Thus, we observed, that this method is not suitable in practice even though similar method is claimed to be used in (Burke and Curtois, 2014).

3.3.2 Mixed-Integer Linear Programming

The problem (3.1) can be modeled by following MIP (mixed-integer linear mathematical program)

$$\min_{\mathbf{x},\mathbf{s},\boldsymbol{\epsilon}} \quad -\sum_{m\in\mathcal{M}}\sum_{k\in\mathcal{K}}(\pi_{mk}-p_{imk})x_{mk}+\sum_{l\in\mathcal{P}_i} c_l^{ub}\epsilon_l^{ub}+c_l^{lb}\epsilon_l^{lb} \tag{3.4}$$

subject to

$$\forall m\in\mathcal{M}: \quad \sum_{k\in\mathcal{K}} x_{mk} = 1 \tag{3.5}$$

$$\forall l\in\mathcal{P}_i, I\in F(l): \quad \sum_{m=I}^{I+|l|-1}\sum_{k\in l(m)} x_{mk}-|l|+1 \le s_{l,I} \tag{3.6}$$

$$\forall l\in\mathcal{P}_i: \quad \sum_{I\in F(l)} s_{l,I}-\epsilon_l^{-} \le ub_l \tag{3.7}$$

$$\forall l\in\mathcal{P}_i: \quad \sum_{I\in F(l)} s_{l,I}+\epsilon_l^{+} \ge lb_l \tag{3.8}$$

$$\forall m\in\mathcal{M}, \forall k\in\mathcal{K}: \quad x_{mk}\in \{0,1\} \tag{3.9}$$

$$\forall l\in\mathcal{P}_i, I\in F(l): \quad s_{l,I}\in \{0,1\} \tag{3.10}$$

$$\forall l\in\mathcal{P}_i: \quad \epsilon_l^{\pm}\ge 0 \tag{3.11}$$

where π_{mk} are the dual multipliers taken from the dual solution of the restricted master problem (see Figure 2.5), the p_{imk} is penalty for not assigning shift k on day m, c_l^{ub} (c_l^{lb}) is a cost of violation of maximal (minimal respectively) matching of a pattern l, $F(l)$ is a set of indicies of feasible matching starts of the pattern l, $l(m)$ is a set of shifts which can take place on day m in pattern l, lb_l (ub_l) is a minimum (maximum respectively) number of matches for pattern l and $|l|$ stands for the length of a pattern l.

Compare objective (3.4) to the pricing problem definition (3.1). The term γ_i is dropped out of the objective since it is a constant. Soft cost penalty c_{il} computation is done by introducing additional binary variables $s_{l,I}$ for each matching position I of the pattern l. These variables are constrained in such a way, that they are taking value 1 if the pattern l is matched starting from position I. They are summed up and compared to the allowed bounds. If a bound is violated, it increases the objective via $\epsilon_l^{\pm}$ slack variable.

For the majority of constraints which are posed in the benchmark instances this model is complete. However, it misses e.g. constraint for minimal/maximal workload across the schedule or its individual parts. Other thing missing in the formulation above is a quadratic penalty function for the violation of constraints.

Since we deal with integer domain (e.g. there cannot be 2.71 patterns present) we can model the quadratic in terms of MIP using following construction where we introduce new variables z_t and impose constraints in form of

$$\epsilon_l = 0z_0 + 1z_1 + 2z_2 + \ldots = \sum_t t z_t$$
$$\sum_t z_t = 1, z_t \in \{0, 1\}$$

and put to the objective scalar multiplication of the vector $\mathbf{z}$ with the designed cost vector $\mathbf{c} = [0\,1\,4\,9\,16\,\ldots]^T$. Note that one needs an upper bound on the number of violations in advance because there must be a finite number of z_t variables.

The main disadvantage of the (3.4) model is that the number of variables grows fast (but it is still polynomial) with the number of patterns present in the problem ($s_{l,I}$ variables).

Overall, MIP seems as a good way to solving the pricing problem. In our case we deal with the $\mathcal{NP}$-hard problem without any obvious structure, thus we have essentially only the branch and bound method to search this space. Since the LP relaxation is very strong (see Section 3.5), we have also very good lower bound on the integer objective, thus, the branch and cut implemented in IP solver is effective solution method.

3.3.3 Informed state space search

The problem of finding the working schedule for the individual employee can be viewed as a state-space search problem. Each state is a complete or partial schedule. The actions we consider are extensions of the partial schedule (one for each shift type). Applicability of an action in a state is given by the branching constraints (see Chapter 4.5). Moreover, there is a cost function associated with each state. The initial state is an empty schedule with the cost of 0 and a goal state(s) are the ones which are complete schedules with minimum value of the cost function. Thus the search space is finite, but very large.

With this settings it essentially coincides with the branch and bound method. We used the informed search method — A* algorithm (Russell et al., 1995). It is the branch and bound like method which specifies the order of the nodes in which they are expanded. We designed an *admissible* heuristics for that (in context of branch and bound framework it is called a lower bound). It is computed as

$$h(s) = \sum_{m=s}^{n} \max_{k \in \mathcal{K}} \pi_{mk} + \sum_{l \in \mathcal{P}_i} c_l^{ub} \max\{0, \#l - ub_l\} \tag{3.12}$$

Since the large number of partial solutions must be evaluated, we have implemented the *delta evaluation scheme* — it allows us to evaluate pattern matching only at the last position rather than in whole schedule. Since the evaluation is the most time consuming operation in this pricing algorithm it leads to the significant speedup.

Although computation of $h(s)$ is fast, it is obviously much weaker in contrast to the linear programming relaxation of (3.4). Thus, this approach pays off only for small scheduling horizons. We found out that A* is able to outperform the MIP based pricing in practice for horizon length $n \leq 14$ (two weeks).

3.3.4 Constraint programming

Constraint programming (CP) is more suitable for testing feasibility rather than optimizing. Thus, we have started by designing a CP model for finding columns without any violation of soft constraints to see how it will perform.

The model is following

$$\forall m \in \mathcal{M} : \quad X_m \quad \in \quad \mathcal{K} \tag{3.13}$$

$$\forall m \in \{\mathcal{M} \mid m \bmod 7 = 5 \vee m \bmod 7 = 6\} : \quad (X_m > 0) \quad \iff \quad B_m \tag{3.14}$$

$$aggregate(\mathbf{B}) \quad \leq \quad ub \tag{3.15}$$

$$\texttt{global\_cardinality}(k_0 - KNum_0, \ldots, k_{|\mathcal{K}|} - KNum_{|\mathcal{K}|}) \tag{3.16}$$

$$\forall k \in \mathcal{K} : \ KNum_k \in \{lb_k \ldots ub_k\} \tag{3.17}$$

$$\forall l \in \mathcal{P}_i : \ \texttt{automaton}(\mathbf{X}, \mathcal{L}_l) \tag{3.18}$$

where X_m is a decision variable assigning shift on day m. The $\mathcal{L}_l$ is a regular language defined by the pattern $l \in \mathcal{P}_i$. The global constraint `automaton` ensures that X is accepted by an automaton equivalent with the language $\mathcal{L}_l$ (see Figure 3.5 for an example of such language).

We further used `global_cardinality` constraint to force assignment of a certain number of each shifts in the schedule (since this is common constraint) in order to strengthen the formulation. Bounds on the occurrence for each of them can be modeled by a single number (the case when desired minimum equals to maximum) or just using additional variable (in our case $KNum_i$ with appropriate domain).

The constraint (3.14) is so called *reified* constraint. The B_m is a boolean variable taking value of 1 if and only if the X_m is not assigned to the *Off* shift. Using B_m variables and

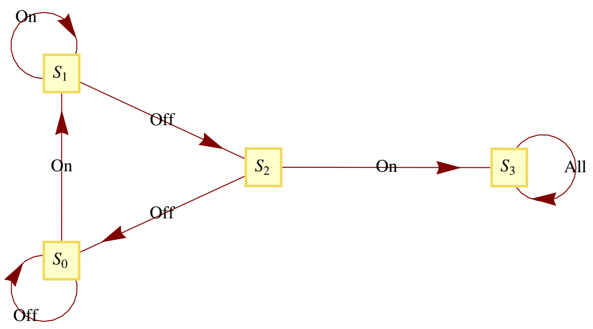


Figure 3.5: Example of an automaton rejecting *On-Off-On* pattern. Initial state is S_0, accepting states are $S_{0...2}$.

appropriate *aggregate* function we can model the requirements for working shifts that can occur during the weekends. Once again, the purpose of these additional constraints is to improve the strength of the model.

It is possible to further extend the model into the form where we state constraints

$$\forall m \in \mathcal{M} : \texttt{element}(X_m, \boldsymbol{\pi}_m, Y_m) \tag{3.19}$$

$$-\gamma_i - \sum_{m \in \mathcal{M}} Y_m + \sum_{l \in \mathcal{P}_i} c_l < 0 \tag{3.20}$$

where `element` constraint ensures that Y_m takes X_m-th value of $\boldsymbol{\pi}_m$. The c_l is a penalty calculated by `automaton` constraint. These additional constraints can be used to test whether the column with the negative cost exists, i.e. to check whether the column generation has converged. However, as we will show below, even with these expressive constraints the model's performance is not impressive.

Computational results Value/variable ordering does matter. We observed that fail-first principle on the most constrained variable with descending value ordering is on average superior approach for solving our model. See detailed results in Table 3.1. Variable ordering: *leftmost* selects the leftmost variable first, *min* (*max*) selects the leftmost variable with the smallest lower (greatest upper) bound, *ff* selects the leftmost variable with the smallest domain, *ffc* selects the variable with the smallest domain, ties are breaked by the most constrainted first. Value ordering: *up* explores domain in ascending order and *down* in descending order. For each combination of variable/values ordering Table 3.1 reports number of backtracks needed to to solve the problem and the overall running time.

During the tests we have used multiple `automaton` constraints — one for each pattern. We have also tried to do the intersection of individual languages and then minimize resulting

	Backtracks/Runtime [-/ms]									
	leftmost		min		max		ff		ffc	
Instance	up	down	up	down	up	down	up	down	up	down
Azaiez	3104/210	4/30	549/70	4/30	2208/260	4/30	3104/230	4/40	270/70	0/30
Millar	1/20	18/50	1/50	14/40	1/20	2/40	1/40	18/20	2/20	25/20

Table 3.1: Computational results of constraint programming model for some instances. First entry of each cell corresponds to the number of backtracks during the solution and the second one shows the running time.

automaton. However, the results were actually slightly worse than when we used separate constraints.

We did not perform measurements for all test instances, since based on preliminary results in Table 3.1, we were not impressed by the power of the CP on this problem enough to continue research in this direction.

3.4 Heuristic methods

It is not necessary to find optimal solution for the pricing problem and prove it in every iteration of column generation. Finding column with the largest negative reduced cost does not guarantee that it will enter the basis, it is just more likely to do so. This means that during *some* iterations of column generation it is not necessary to find optimal column with respect to the definition (3.4), but even just some good-quality solutions with negative reduced cost could do the same job as the optimal solution. On the other hand, one has to keep in mind that column with *positive* reduced cost *cannot* improve the objective of the primal problem (at least not immediately in current iteration).

Essentially, the only moment when it is necessary to run the exact algorithm is the stage, when the proof is needed for the master model being solved up to the optimality (e.g. before branching, for obtaining a lower bound etc.).

3.4.1 Heuristic MIP

We can easily turn our MIP formulation for pricing problem into an effective heuristics. We set parameters for IP solver to solve the problem up to the specified gap, i.e. proportion between lower bound and upper bound (usually something from 0.5% to 15%). Moreover, we update gap parameter dynamically based on the speed of convergence of the primal objective. If the convergence is fast, it is presumably sufficient to solve problems heuristically. When the convergence is becoming slow, it starts decreasing the gap. When the column generation almost converged, it sets gap to zero, therefore it turns into the exact method once again.

More detailed treatment of the heuristic MIP pricing can be found in Chapter 6.3.

3.5 Lower bound

An obvious lower bound on the reduced cost (3.1) is

$$lb = -\gamma_i - \sum_{m \in \mathcal{M}} \max_{k \in \mathcal{K}} \pi_{mk} \tag{3.21}$$

since $\forall \mathbf{x} \in \{0,1\}^{|\mathcal{M}| \times |\mathcal{K}|} : c_{il}(\mathbf{x}) \geq 0$. When one constructs partial solutions, e.g. left-to-right as our A^* does it, we can strengthen this bound based on the pattern matches which are over their upper bound ub_l. However, in practice, this bound is relatively weak.

The other way for obtaining the bound is to deal with the whole problem at once. When relaxing on integer property of constraints in formulation (3.4), the new problem can be used as a lower bound for pricing problem. This is easy to see since we are minimizing over the larger set, thus objective value in the optimum will not be greater than the true value of (3.1). The advantage of linear relaxation of (3.4) is that it gives us polynomial time computable bound in contrast to the original $\mathcal{NP}$-hard task.

We empirically observed, that the power of linear relaxation of the model (3.4) is very good, i.e. on average we obtain values about 5% lesser than the integer optimum (see Figure 3.6). One can see that linear relaxation tends to be weaker at the end of the column generation. In our opinion, this is caused by the fact, that it gets harder to find columns with negative reduced cost at the end of the process. Therefore, more binary variables in the MIP model are used, which causes troubles for relaxation.

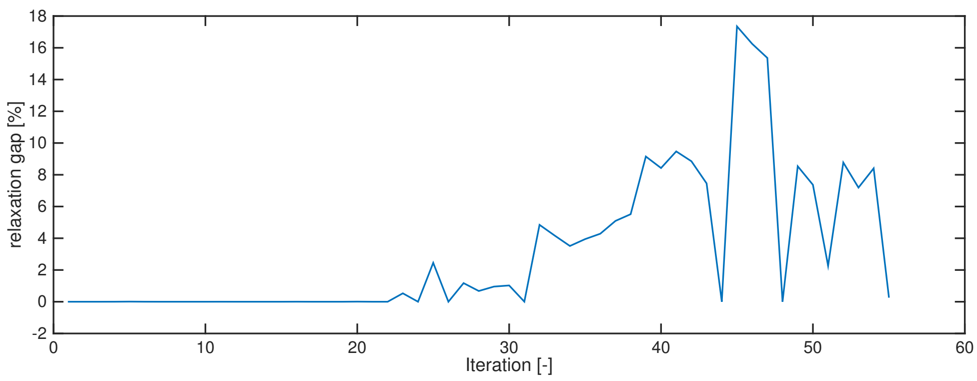


Figure 3.6: LP relaxation is tight. Example gap between IP optimum and LP solution for Azaiez instance.

It is worth it to note, that this bound can be also used as an admissible heuristics for the informed search described in Chapter 3.3.3.

3.5.1 Tighter bound via Semidefinite Programming

Different way how to obtain a relaxation is to formulate the problem as a *Semidefinite Programming Problem* (SDP) — it is a special case of convex optimization problem which is more general than the linear and quadratic programming problems. In fact, they are just special cases of SDP. In paper (Alizadeh, 1995, pp. 25-27) the method of SDP relaxation is described, which in theory gives us tighter relaxation of 0-1 linear programming problem than regular LP, i.e. it creates non-linear convex hull of feasible points which is closer to the integer hull of the original problem. Importantly enough, a SDP problem can be solved in polynomial time like linear programming.

The particular example is *weighted maximum stable set* problem, where we are asked to pack a set of vertices of a graph which give us the highest utility given that we are forbidden to select both vertices connected by an edge. It can be proved, that SDP relaxation of the IP formulation of the problem already contains some classes of specific cuts such as *clique cuts*, *odd hole cuts* and others implicitly which are important in order to solve original IP problem fast.

However, considering the power of linear relaxation, there is arguably not so much potential for SDP relaxation to be useful in our problem since SDP relaxation's size grows with the square of the number of constraints in the IP formulation.

3.6 Upper bound

Since we are interested in columns with negative reduced cost only, we have an obvious upper bound on the reduced cost (3.1). When a constant term is dropped out of the objective as in model (3.4), the upper bound became the γ_i term.

Even though obtaining γ_i at each iteration is for free, it is not a very tight bound, especially from the beginning of the column generation. We will address this issue in Chapter 5.2.

3.7 Additional remarks

Cutting planes in dual master problem Pricing problem, in general, can be accelerated by putting additional valid inequalities into the dual formulation of the master problem. This would reduce domains of dual variables and consequently could improve column generation convergence. However, in order to preserve optimality of the branch and price algorithm, one has to assure that putting a cutting plane into the dual preserves at least one optimal solution of the pricing problem.

The *stabilization* of the column generation can be seen as a form of dual cutting planes, i.e. it restricts the domains of the $\boldsymbol{\pi}$ variables. We will discuss it in details in Chapter 6.2.1.

Single skill When we recall the equation (3.1) defining the pricing problem, it can be seen that we are using π_{mk} variables although the dual master model (Figure 2.4) contains π_{mjk} variables. The reason is that we have assumed, that every employee has exactly one skill, thus for the given subproblem (finding a schedule for single employee) j is fixed. This holds for most of the benchmark instances (Curtois, 2014).

4 Master problem

Under our decomposition scheme, the master problem takes care of coupling employees' schedules into the complete roster. It is done in a way, such that the objective is minimal. The objective combines penalties for the coverage requirements and costs of individual schedules.

Solution of restricted master problem provides dual multiplies for the pricing problem. It generates a new column, which could enter the basis of the restricted master problem. If it does, we obtain different solution, hence the new column can be constructed.

This chapter is organized as follows. In Chapter 4.1 we define the master problem, in Chapter 4.2 we discuss ways to obtaining initial solution, then in Chapter 4.3 properties of lower bound provided by the master problem are described. In Chapter 4.4 we show usage of Lagrangian relaxation, Chapter 4.5 deals with branching mechanisms and in Chapter 4.6 we conclude with some improvements for obtaining the upper bounds.

4.1 Definition

The master problem takes form of *set covering polytope* (Barnhart et al., 1998). Given some set of variables, their costs and constraints on them, it selects a feasible subset of them with the minimal sum of their costs and penalty for constraint violations.

Master problem is modeled by the linear program in Figure 2.3. By examining its constraints it is clear, that it cannot be infeasible unless there is an employee with empty set $\mathcal{F}_i$. Model's abilities also include 4-piecewise linear penalty function for staffing requirements (preferred and both minimal/maximal coverages penalties with different slopes) for each day and shift type (see Figure 4.1). These coverage requirements can be specified across subsets of skill competencies independently.

Moreover, it allows employees to express their preferences for each

- shift on/off assignment
- day on/off assignment

with individual weights. The particular day-shift assignment can be also *frozen* (i.e. hard constrained assignment).

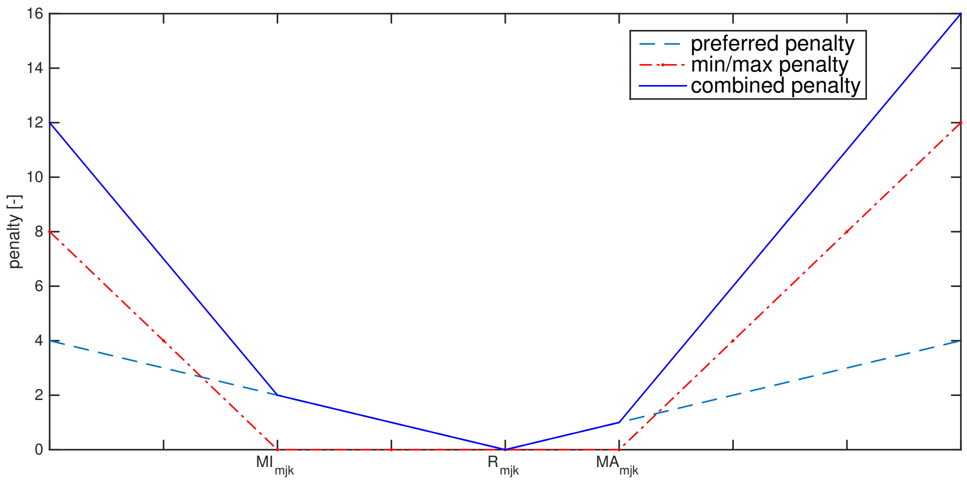


Figure 4.1: An example of coverage penalties.

4.2 Initial solution

In order to start the column generation, the master problem must be supplied with an initial feasible solution. In the master problem (2.21) it is rather easy. Any set of columns will do (until there is at least one for each employee) since we consider nearly all constraints as the soft ones. However, completely random initial solution is not usually desired.

From the dual space point of view, the initiation heuristics serves, after the first resolving of the restricted master, as an estimate of the optimal dual prices $\boldsymbol{\pi}$. When the initiation solution does not cut out the dual feasibility polyhedron sufficiently enough, more iterations of column generation will be needed in order to find the optimum of the master problem.

On the other hand, an inappropriate initial solution can guide the column generation to the "local optima" which may be hard to escape, i.e. consecutive dual solutions will be far from the optimal ones, thus the pricing problem will cut out unnecessary parts of feasibility region repeatedly.

4.2.1 Single-pass heuristics

An initial solution can be obtained by the pricing problem solver. We proposed the following method. It randomly selects the first employee and generates a schedule for him/her with the minimum soft cost violation with respect to him/her working preferences. For that, it uses the same pricing solver as in column generation procedure but with zero dual prices for assignments constraints. Then, it iteratively solves pricing problem for all employees using dual prices modified by the information about required coverages and assignments already made in previous steps (see Figure 4.2). The equations generating a column for employee $t+1$ are based on the state of the partial roster created up to employee t. They are stated as

$$\gamma_i^{(t+1)} \leftarrow +\infty \tag{4.1}$$

$$\pi_{mjk}^{(t+1)} \leftarrow \begin{cases} 0 & \text{if } k = 0 \\ \frac{c^u_{mjk}}{|\mathcal{E}|-t}(R_{mjk} - R^{(t)}_{mjk}) & \text{if } R^{(t)}_{mjk} < R_{mjk} \text{ and } k > 0 \\ -c^o_{mjk}(R^{(t)}_{mjk} - R_{mjk} + 1) & \text{otherwise} \end{cases} \tag{4.2}$$

The assignment $\pi_{mj0} = 0$ is made since we allow to take day off without penalization because no penalties for overstaffing/understaffing at *day off* are imposed (we consider the off shift as $k = 0$). However, when coverages $R^{(t)}_{mjk}$, $k > 0$ of partial roster are under the desired level, we increase importance of those shifts proportionally to the penalty of over staffing and employees left for allocation. The intuition is that if some coverages are not yet satisfied but we still have many employees to assign, it is not an issue because some of the unscheduled employees will cover those shifts in next steps. On the other hand, if some preferred coverages are already fulfilled, we will adjust our preference to not to place shifts there by the factor of over staff penalty.

When the problem description does not provide a preferred coverage R_{mjk} but only minimal and maximal ones, we can use them directly in updating equations (4.1) and (4.2) or just estimate the "preferred" coverage as average of both of them.

As we mentioned, we use pricing solver for deriving columns for the initial solution. When the MIP based pricing solver is employed, we do not solve problem up to the optimality, but with specified gap of 5% (see Chapter 3.4.1).

4.2.2 Pattern pump heuristics

We proposed an initial heuristics that generates all schedules for employees with given working contract with desired soft cost (e.g. zero). It creates a master model using these columns which we assign to all employees with the appropriate working contract and solve it resulting restricted master model as an IP. We essentially "pump" the patterns inside to the master model, hence the name *Pattern pump* heuristics.

To find these columns with zero costs for violation of soft constraints we use the same A^* algorithm as which can be used in pricing problem (see Chapter 3.3.3). For smaller problems (i.e. 2 weeks scheduling horizon, 3 shift types), generation of these patterns is very simple since we can prune the nodes (partial schedules) as soon as some of these constraints are violated.

$$\begin{array}{cccccccc}
\hline
1 & 0 & \cdots & 1 & \xrightarrow{\Sigma} & R^{(t)}_{100} & \rightarrow & \pi_{100} \\
0 & 1 & \cdots & 0 & \xrightarrow{\Sigma} & R^{(t)}_{101} & \rightarrow & \pi_{101} \\
0 & 0 & \cdots & 0 & \xrightarrow{\Sigma} & R^{(t)}_{102} & \rightarrow & \pi_{102} \\
1 & 0 & \cdots & 1 & \xrightarrow{\Sigma} & R^{(t)}_{200} & \rightarrow & \pi_{200} \\
0 & 1 & \cdots & 0 & \xrightarrow{\Sigma} & R^{(t)}_{201} & \rightarrow & \pi_{201} \\
0 & 0 & \cdots & 0 & \xrightarrow{\Sigma} & R^{(t)}_{202} & \rightarrow & \pi_{202} \\
\vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\
1 & 1 & \cdots & 0 & \xrightarrow{\Sigma} & R^{(t)}_{mjk} & \rightarrow & \pi_{mjk} \\
\hline
(1) & (2) & \cdots & (t) & & & & \boldsymbol{\pi}^{(t+1)} \\
a_{11} & a_{21} & \cdots & a_{t1} & & & &
\end{array}$$

Figure 4.2: An example of a single-pass initial heuristics. A new column (t+1) is created based on the dual multipliers calculated by coverages $R^{(t)}_{mjk}$ satisfied up to (t).

The nice property of formulation (2.21) is its strong LP relaxation thus IP solver is usually able to solve the problems with thousands of variables in the root node (i.e. without branching) just with cutting planes. Integer solution from the IP solver then serves as an initial solution.

The advantage of the pattern pump heuristics is that it provides good upper bounds in a few milliseconds. The drawback is that it works in reasonable time only for smaller instances (i.e. where columns can be enumerated fast, as stated above).

4.3 Lower bound

When the master problem is solved up to the optimality, its objective value serves as a lower bound on the objective value of the whole (integer) problem specified by the original formulation (see Chapter 2.2).

These bounds are tight in practice (Burke and Curtois, 2014) — on the majority of test instances it is within gap less than 1%. However, if it is not the case, we can sometimes deduce stronger bounds by employing the condition of being integer-valued.

We found a way how to improve lower bounds. The point is that when the optimal objective of the relaxed problem (optimal objective value of the master problem) is greater than 0, then we know that at least one soft constraint is violated. By looking at all the penalties we tight up bound to the smallest of them as

$$lb = \max\{z_{rmp}, \min_{l,m,j,k}\{\text{penalty for pattern}\, l,\ \text{penalty for coverage constraints}\, m, j, k\}\} \quad (4.3)$$

The z_{rmp} denotes the optimal objective value of the master problem (not necessary integer solution). Keep in mind that above is possible only in case that $z_{rmp} > 0$.

Consider instance *Valouxis* — column generation in root node yields a lower bound 8. However, since all penalties for coverage constraints and costs of patterns are greater than 8, we can tight up the lower bound according (4.3) which in the case of *Valouxis* instance is 20, i.e. optimal value. This technique was not mentioned in either work by (Maenhout and Vanhoucke, 2010) or (Burke and Curtois, 2014).

Sometimes it would be convenient to have some lower bound even before master problem is solved up to the optimality. It can be done by *Lagrangian relaxation*, which let us compute the lower bound at the arbitrary iteration of column generation.

4.4 Lagrangian relaxation

We can use column generation for calculating a lower bound on the value of master problem even before it converged. When all subproblems (employees) are solved up to the optimality it can be shown (van den Akker et al., 2002) that following equation always holds

$$\hat{z} \geq z_{rmp} + \sum_{i \in \mathcal{E}} \mu_i \tag{4.4}$$

where $\hat{z}$ is optimal value of master problem, z_{rmp} is current value of restricted master problem and μ_i is reduced cost for i-th pricing problem.

Figure 4.3 depicts the lower bound obtained by equation (4.4). Lower bound does not have to necessarily develop in monotonic way (it can decrease between iterations). However, since the equation (4.4) holds in every iteration, one can store the maximum of it and use it as the lower bound.

Lagrangian relaxation can be also used to address one of the issues in column generation procedure — so called *tailing off effect*. This is a phenomenon where by the end of column generation a large number of columns with only a little negative reduced cost is generated. Thus, the algorithm could spend a majority of runtime in this tail trying to solve the relaxation of the original problem. However, when the solution of the relaxation becomes difficult, it is not desirable to spend too much time there.

It can be shown (Barnhart et al., 1998) that when following inequality holds for any fixed dual solution $\boldsymbol{\gamma}$ and $\boldsymbol{\pi}$

$$z - \lfloor z \rfloor > -\sum_{i \in \mathcal{E}} \mu_i \tag{4.5}$$

where μ_i is the reduced cost (see Chapter 2.6.1 for its calculation) of the optimal solution for subproblem i, and z is the objective value of restricted master problem, then we can stop

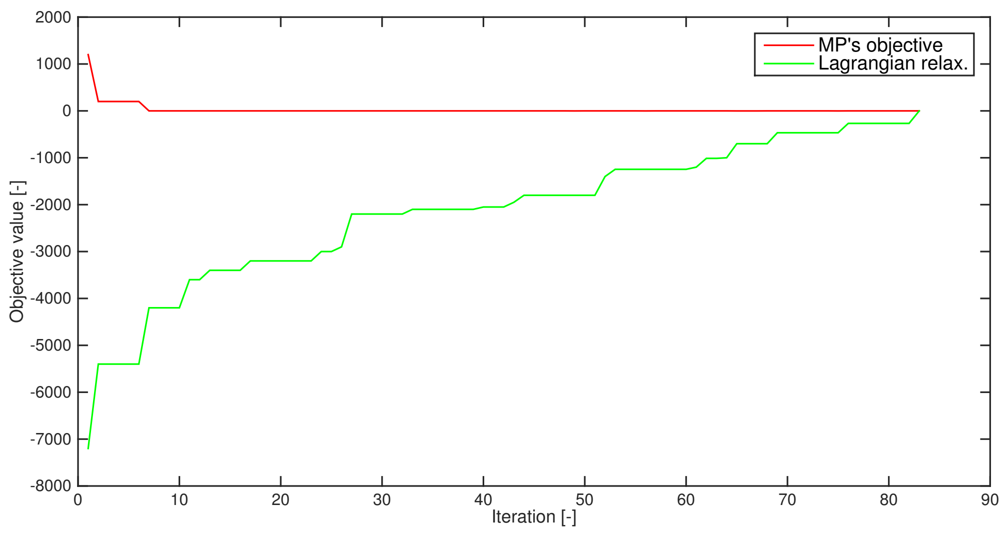


Figure 4.3: Lower bound computed by Lagrangian relaxation at each iteration of column generation in *Millar* instance.

column generation procedure without taking a risk of missing the optimal solution.

This result follows from the fact that we are working with integer-valued objective. Informally, the reduced cost can be seen as the maximal amount of the primal objective that could be improved if it would enter the basis. When the potential improvement given by all subproblems is less than 1, the column generation has converged.

4.5 Branching methods

Branching in general is the method which ensures integer property of a solution. In a fractional solution, it selects a variable and value for it, fixes them and resolves the problem. This procedure is repeated until the solution is integer. The integer solution then serves as an upper bound. The optimal solution of the whole problem is the one with the lowest objective value among the integer ones.

Branching is a complete method since every time it branches it splits solution space into disjoint parts in such a way, that no integer solution is omitted. Branching itself can be time exhausting since, in the worst case, it basically enumerates all the solutions (and there is exponential number of them). However, in practice, it is not the case. We may prune many nodes using lower bounds obtained by a column generation procedure which are very tight so these nodes are not visited.

In order to perform branching we have to select a variable and decide which values it takes. We will discuss both these decisions in the following paragraphs.

4.5.1 Branching on master variables

Probably the most natural variables to branch on are the ones that specifies selected columns (i.e. y_{il} variables in (2.21)). Thus, branching on master variables makes decision like *employee i is assigned to pattern l* and *employee i is not assigned to pattern l.* The former could serve for obtaining the integer upper bound, but the latter says nearly nothing — the new problem is the same as its parent with the exception that only a single pattern for one employee is forbidden.

4.5.2 Branching on original variables

The other reasonable choice of variables to branch on are original ones — these assign an employee to a specific shift on specific day. The branching decisions then looks like *employee i is assigned to a morning shift on day j* and *employee i is forbidden to work on morning shift on day j.* Moreover, since an employee is always assigned to the exactly one shift at each day (it is one of few hard constraints in our problem) it can also forbids other assignments in the branch, where employee is assigned to the particular shift in a positive sense (see Figure 4.4).

Those decisions are usually better than the ones made on the master variables (Maenhout and Vanhoucke, 2010). Following sections will therefore focus on the branching on the original variables.

It is important to note that those branching decisions change the structure of the pricing problem and the algorithm solving it has to take it into account. Essentially, branching on original variables splits the search space based on whether the some shift is allocated at some day for an employee. This creates a new master problem, which is then solved by column generation, thus the pricing problem is solved in it. However, this master problem is different from its parent and the pricing problem must obey this constraint. It is introduced to the pricing problem by constraints in form of $x_{mk} = 1$ or $x_{mk} = 0$.

4.5.3 Variable selection

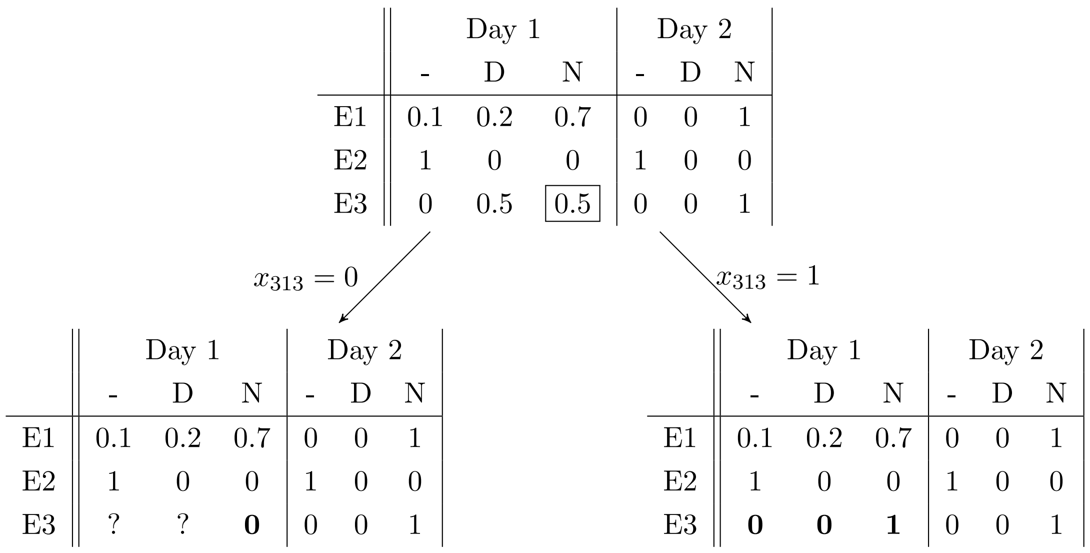
Before we branch we have to choose variable(s) to branch on. The general idea is that we want to select a variable and value for it in order to resolve the main cause of not being integer-valued. Fractional value for some variable appears when there is a conflict caused by the linear relaxation in assignment for given shift, i.e. should an employee be assigned on given shift or not?

Most fractional value Selects a variable with the most fractional value. Since we consider values between 0 and 1, it selects the variable with value closest to 0.5. In case of ties it can select e.g. the one which influences the more expensive coverage constraint. It tries to resolve the most conflicting assignments.

Closest to one Selects a variable with the value closest to 1. It resolves *almost* decided assignments. However, in order to preserve completeness of the algorithm, it also has to consider the branch with opposite decision.

4.5.4 0–1 branching

The most obvious method for branching on the original variables is to use 0–1 branching. It simply selects a variable and creates two branches, where for each it fixes the decision. An example is depicted in Figure 4.4.



	Day 1			Day 2		
	-	D	N	-	D	N
E1	0.1	0.2	0.7	0	0	1
E2	1	0	0	1	0	0
E3	0	0.5	$\boxed{0.5}$	0	0	1

$x_{313} = 0$

	Day 1			Day 2		
	-	D	N	-	D	N
E1	0.1	0.2	0.7	0	0	1
E2	1	0	0	1	0	0
E3	?	?	**0**	0	0	1

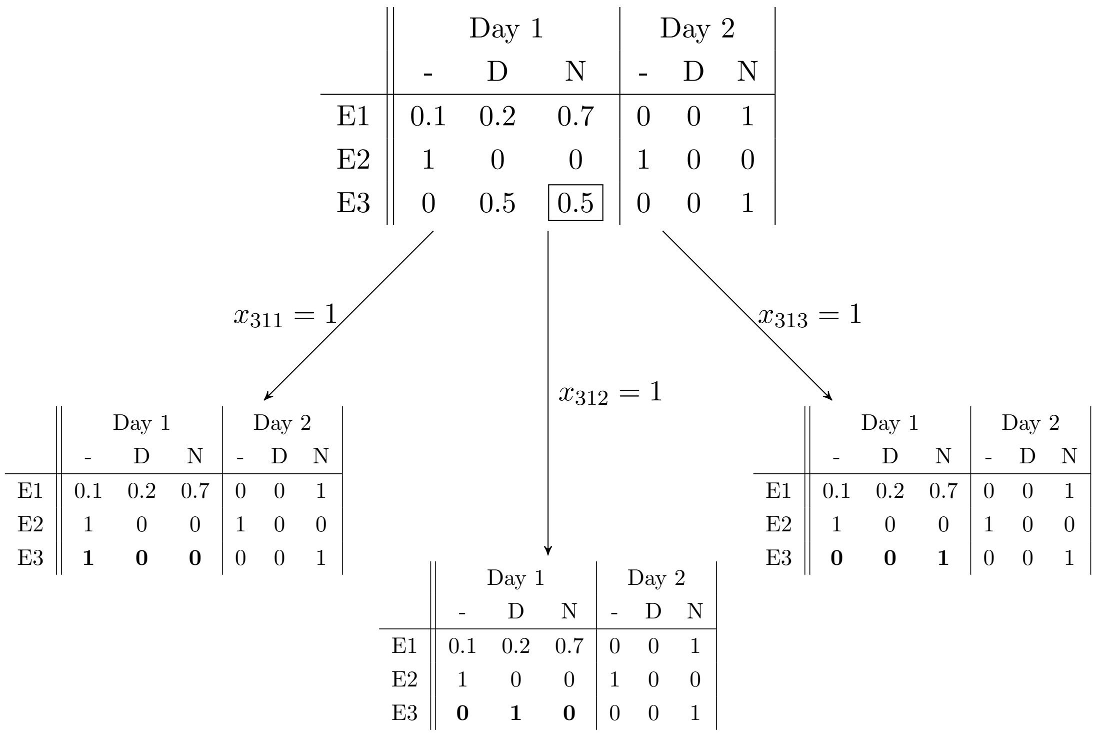
$x_{313} = 1$

	Day 1			Day 2		
	-	D	N	-	D	N
E1	0.1	0.2	0.7	0	0	1
E2	1	0	0	1	0	0
E3	**0**	**0**	**1**	0	0	1

Figure 4.4: Example of 0–1 branching with the most fractional value variable selection. It creates exactly two branches — in the first one it forbids the assignment and in the second one it fixes the shift for a specific day and employee (and consequently forbids other shifts for the same day). Fixed assignments are in bold.

4.5.5 1/../S branching

This method works in quite a similar way as the 0–1 branching but it creates branches for all possible shift assignments for a given day. When there are only 2 possible shifts per day it coincides with the 0–1 branching. See Figure 4.5 for an example.



	Day 1			Day 2		
	-	D	N	-	D	N
E1	0.1	0.2	0.7	0	0	1
E2	1	0	0	1	0	0
E3	0	0.5	$\boxed{0.5}$	0	0	1

$x_{311} = 1$

	Day 1			Day 2		
	-	D	N	-	D	N
E1	0.1	0.2	0.7	0	0	1
E2	1	0	0	1	0	0
E3	**1**	**0**	**0**	0	0	1

$x_{312} = 1$

	Day 1			Day 2		
	-	D	N	-	D	N
E1	0.1	0.2	0.7	0	0	1
E2	1	0	0	1	0	0
E3	**0**	**1**	**0**	0	0	1

$x_{313} = 1$

	Day 1			Day 2		
	-	D	N	-	D	N
E1	0.1	0.2	0.7	0	0	1
E2	1	0	0	1	0	0
E3	**0**	**0**	**1**	0	0	1

Figure 4.5: Example of 1/../S branching. For each possible shift assignment for a given day it creates a new branch with fixed value. Fixed assignments are in bold.

4.5.6 Constraint branching

Constraint branching imposes constraints over multiple original variables. (Maenhout and Vanhoucke, 2010) used constraint branching for ensuring whether an employee serves same shifts on consecutive days. The reason behind this choice is hidden in their constraint set, where the constraint *minimum number of consecutive days* was used. Constraints used by the branching rule then looks like

$$x_{imjk} + x_{i(m+1)jk} = 2$$

where it e.g. requires that employee i is assigned on the day m and $m + 1$ to the shift k. However, this leads to the unbalanced search tree, since the other branch is $x_{imjk}+x_{i(m+1)jk} \neq 2$, which covers 3 distinct options for variable's values to take. Therefore, this subtree contains problem similar to its parent.

Similar branching method can be used if the problem poses constraints on weekend assignments, i.e. the requirement whether the employee serves over the weekends. It is reasonable to expect that most of these requirements are satisfied in high-quality solutions.

These more complicated branching schemes can be seen as a form of *look ahead* techniques known from the Constraint Programming. It tries to resolve variable's values in advance using constraints in an *active way*.

Although constraint branching may reduce the path to the solution, it pays for it by creating unbalanced search tree. (Maenhout and Vanhoucke, 2010) observed, that on average, it didn't pay off for their test instances.

4.5.7 Strong branching

Strong branching is general branching paradigm which selects variable to branch on based on an estimate of how the branch could improve the objective value. Typical choice is to evaluate the LP relaxation with every variable fixed and selects e.g. the one that yielded with the lowest objective. Obviously, strong branching is more time demanding than ordinary branching schemes (e.g. most fractional variable).

There can be variety of evaluation functions for strong branching. Some of them are based on the LP relaxation value, other are based on the measure of fractionality of LP solutions. This obviously hints to the *entropy* measure. In paper by (Gilpin and Sandholm, 2011) was shown, how information-theoretic measures such as entropy can help to solve difficult real-world MIP problems.

4.5.8 Impact to performance

If one deals with solving problems optimally, (Barnhart et al., 1998) suggest to use branching scheme which creates balanced search tree. When the focus is rather on finding high-quality solution quickly, then some heuristic branching schemes can be employed (e.g. rounding up some assignments in fractional solutions) which creates unbalanced search tree or even prevents from backtracking.

The work (Maenhout and Vanhoucke, 2010) explored branching schemes we have described above. Their result was that superior strategy is 0–1 branching with selection of the most fractional variable. On the other hand (Burke and Curtois, 2014) reports, that the choice of the branching strategy has no significant impact on the performance. The reason is that (Maenhout and Vanhoucke, 2010) used custom generated instances coming from the same distribution of parameters while (Burke and Curtois, 2014) used real-world instances from various hospital environments.

Our conclusion are the same as (Burke and Curtois, 2014), thus we used 0–1 branching with the most fractional variable selection.

4.6 Upper bound

Upper bound on the integer objective might be obtained in following ways:

1. from the initiation heuristics
2. at the every iteration of column generation
3. primal heuristics — resolve master problem as an integer program

Ad. 1. When we have a heuristics that generates exactly one column for each employee, we have obviously an upper bound (see Chapter 4.2).

Ad. 2. Moreover, the solution of restricted master problem may serve also as an upper bound whether the solution of it is integer-valued (the integer property of the solution is not guaranteed by column generation, see Chapter 2.6).

Ad. 3. However, we can enforce the integer property. Having a set of columns, one can construct a model in Figure 2.3 with integer constraints on $\mathbf{y}$ variables. Solution of this modified problem gives us an integer solution which can serve as an upper bound. Even though this lifts up computational complexity from $\mathcal{P}$ to $\mathcal{NP}$-hard, solving modified problem is not time demanding in most cases. Due to its *set covering* structure, IP solver is able to apply various cuts to speed up the computation. Moreover, we can use additional constraints on the objective value like *objective is greater or equal to the value obtained by linear relaxation* and *objective is less or equal to the value of known upper bound* which helps to prune the search space. Further details are presented in Chapter 6.3.

4.7 Column management

So far, we have talked about how to find a column which could enter the basis. Although in an optimal solution we need only a single column for each employee, we typically generate much more than that. Columns generated from the beginning were optimal with respect to dual prices, which are no longer relevant. Therefore, these old columns are not likely to enter current basis once they left it. Moreover, their current reduced cost may be even greatly positive so they even cannot enter the basis in current iteration.

Since these column are not probably useful, we can remove them from the master model. This will decrease the computational demand for the Simplex algorithm which solves restricted master problem frequently (Lübbecke and Desrosiers, 2005).

Removal of a column cannot affect the optimality of the branch and price algorithm. In the worst case, the columns that were removed will be generated again if it turns out as needed in later iterations.

5 Machine learning methods

In this chapter we will explain how machine learning methods relates to our solution method. In Chapter 5.1 we will discuss properties of our problem and considerations regarding to the application of machine learning methods. In Chapter 5.2 we describe proposed the algorithm for improving upper bound for pricing problem and we show an interpretation of its effect in Chapter 5.2.2 and results in Chapter 5.2.3.

5.1 Motivation

As we have shown in Chapter 2 where the branch and price algorithm was described, it consists of a large number of smaller problems (i.e. pricing problems, for more details see Chapter 3) which are of the same computational complexity as the original problem, i.e. $\mathcal{NP}$-hard. Those problems are solved in a sequence and the solution of each problem influences the next one in the sequence. Moreover, the next problem in the sequence is not trivially known, although it is given in deterministic but complicated way (new column influences the optimal solution of linear programming problem and results in different dual solution). Therefore, the problems are not random ones, but share some common properties.

Currently, algorithms solving these problems are not able to reuse information obtained in the past even though these problems are clearly closely related. Therefore, hundreds and thousands of hard optimization problems are solved from the scratch during the whole run of branch and price algorithm. Clearly, a speedup of these problems has an effect on the overall runtime. However, one has to keep in mind some considerations regarding to reuse of such information extracted in the past when the exactness of the algorithm must be preserved.

So far, we talked only about reusing information gained during the solving of a single master problem (i.e. the whole original problem instance). The obvious question might be, whether it is possible to extend the extraction of good decisions that led to high-quality solutions (or even to the optimal ones) on instance-wise level. Those decisions could be e.g. branching decisions, frequently appearing patterns in the individual schedules, etc. When one consider the real-world benchmark instances (Curtois, 2014), which are in our main interest, he/she will find out that these are highly *heterogeneous* in terms of the number of employees, length of planning horizon, number of shift types, coverage constraints and, the most importantly, working restrictions constraints. Clearly, this makes sense, since these instances are really taken from different hospital environments[5].

[5]This is also the reason that makes our task challenging. It might be easy to design an algorithm solving effectively problems with given constraint set, but we have to address the problem on very general level.

From the statistical point of view, one would say that these instances are not drawn from the identical distribution. Therefore, beside the fact that there are too little instances as data samples, statistical machine learning does not seem as a feasible method to be applied for deriving properties of the problem on the instance-wise level. The main assumption of i.i.d. (independent identically distributed) about samples is clearly violated in such a way that it cannot be even pretended it holds.

We believe, that deriving useful properties of problem instances needs to be done during the runtime (i.e. *online*) specifically for each problem instance. For example, it can be proved that some instances of Nurse Rostering Problem are polynomially solvable[6], therefore a lot easier to solve than expected. It would be useful to derive e.g. symmetries in the problem that can be exploited without sacrificing optimality. However, this is also beyond the abilities of statistical machine learning. To address this, one has to shift from a paradigm of learning by *observations* to the learning by *reasoning* (i.e. deriving logically correct conclusions which are not given explicitly about the problem instance).

In Chapter 5.2 we address the problem mentioned from the beginning of this chapter. We will show how to partially overcome difficulties raised by complexity of our problem instances by designing an online learning algorithm for deriving useful information for the pricing problem. Moreover, such methods can have impact not only on Nurse Rostering Problem studied in this thesis, but also might improve generic branch and price approach as the method for solving large integer problems. Therefore, the method introduced in this chapter has applications even outside of the domain of personnel rostering.

5.2 Improving upper bound in pricing problem

While solving the pricing problem, where we are looking for the schedule for invididual employee, two bounds are commonly used — the lower bound (see Chapter 3.5) and the upper bound (i.e. 0, since we are looking for solutions with negative reduced cost). For branch and bound based solution method it holds (assuming minimization), that each search node (partial individual schedule) has a lower bound associated with it and all the nodes shares the same upper bound. Since we deal with a problem with soft constraints, one of a few ways how to prune nodes in the search tree is to prove that their lower bound has greater value than the best currently known upper bound. These two bounds are getting closer to each other during the run of a pricing algorithm. Naturally, it is desired to have them as close as possible from the start so more nodes can be pruned early.

[6]This is not a surprising fact, since for example the shortest path problem in general is $\mathcal{NP}$-hard, however, special cases without negative circuits can be solved in polynomial time.

Consider the high-level form of the objective function for the pricing problem

$$f(\mathbf{x}) = -\gamma - \boldsymbol{\pi}^T\mathbf{x} + c_{il}(\mathbf{x}) \tag{5.1}$$

where γ and $\boldsymbol{\pi}$ are constants and c_{il} is a function counting soft cost violations. Since c_{il} is not convex in $\mathbf{x}$, it is hard to find the global minimum of f. We want to find $\mathbf{x}$ such that $f(\mathbf{x}) < 0$, thus, we immediately have an upper bound $\gamma > z$ on the $z \equiv -\boldsymbol{\pi}^T\mathbf{x} + c_{il}(\mathbf{x})$. In this chapter we address the question, if it is possible to obtain tighter bound on value of z.

We propose the following online algorithm for predicting the upper bound. At each iteration the algorithm observes new features values and has access to the previous features and their target values. Based on this information the algorithm outputs its new prediction and use it in pricing solver. After the run of the pricing solver, it obtains the actual true value.

5.2.1 Robust regression problem

Since the algorithm predicts values from continuous interval, we face a *regression problem* (Williams and Rasmussen, 2006). Arguably, the most popular and one of the simplest solution methods is *ordinary least squares method* (LSM). This method minimizes the sum of residual squares (l_2 norm of the error vector), where the predictive hypothesis is linear function in the basis variables (features). From the probabilistic point of view, the least square method is maximum likelihood estimate for the case of linear regression model where we assume that the posterior distribution of values takes form of $p(y|\mathbf{x}) = \mathcal{N}(y|\mathbf{w}^T\mathbf{x})$ and samples are drawn independently.

For the upper bound prediction the LSM is not suitable — the loss function is skewed in a sense that we do not want to underestimate the true value of z (what will happen when it is the case will be described later). Moreover, it cannot be expected to have predicted values exactly equal to the target values, we just want to have them in reasonable small distance from it. These requirements led us to develop our custom criterion and the whole algorithm later on. Therefore, it is meaningless to compare achieved criterion values with e.g. LSM, since we will present below a training algorithm that minimizes the empirical risk.

We designed our criterion as the sum of *skewed epsilon insensitivity* loss functions (see Figure 5.1). Additionally, we discount the contribution of older datapoints to the final loss function by logarithmic factor (see Figure 5.2), since more recent observations are more important to us. Moreover, this serves as a regularization and helps to overcome so-called *heading-in effect* (see Chapter 6.2.2).

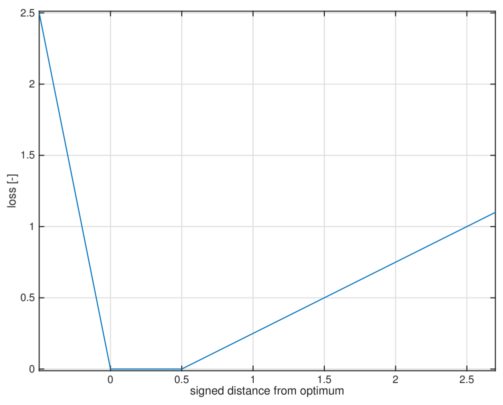


Figure 5.1: Example of loss function for a single datapoint, $\epsilon = 0.5$

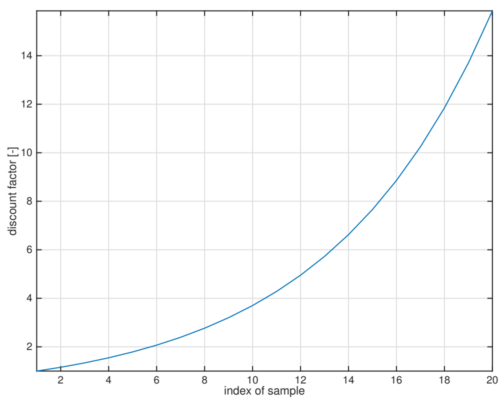


Figure 5.2: Discounting function for previous datapoints

Our model can be trained by the following mathematical program

$$\min_{\mathbf{w},\mathbf{r}} \sum_{i \in \mathcal{D}} c_i^+ r_i^+ + c_i^- \max\{r_i^- - \epsilon, 0\} \tag{5.2}$$

subject to

$$\forall i \in \mathcal{D}: \quad \mathbf{w}^T \mathbf{x}_i + r_i^+ - r_i^- = y_i \tag{5.3}$$

$$\forall i \in \mathcal{D}: \quad r_i^+, r_i^- \geq 0 \tag{5.4}$$

$$\mathbf{w} \in \mathbb{R}^n \tag{5.5}$$

where $c_i^{\pm}$ is discounting constant for datapoint $i \in \mathcal{D}$. Value of $r_i^{\pm}$ measures error made by prediction for datapoint i. Then, the predictive hypothesis is given as $y = \mathbf{w}^T\mathbf{x}$. It can be shown that problem (5.2)–(5.5) can be posed as a linear program.

Moreover, this model has a connection to the Support Vector Machines (SVM) for regression. If we would alter the objective (5.2) by adding the quadratic term $\frac{1}{2}||\mathbf{x}||_2^2$ we would essentially get the maximum margin solution, which is produced by SVMs (Murphy, 2012). However, it would make the training process harder, since it would turn the LP problem to Quadratic Programming (QP) (although still polynomial) problem.

The method described above can be treated as an online learning problem. We developed the algorithm for such setting. It is based on both column generation and cutting planes method. For each new datapoint it adds a variable (column) to the objective function (5.2) and then a cutting plane (constraint) in form of (5.3) to make sure that predicted value is close enough to the true value. (Algorithm 2).

The feature values used for the prediction are an important part to discuss. In pseudocode of the algorithm they are denoted by the mapping $\boldsymbol{\phi}(\gamma_i, \boldsymbol{\pi})$, therefore they are based on the

Algorithm 2: Upper bound prediction for pricing problem

```
1  do
2      $(\gamma_i, \boldsymbol{\pi}) \leftarrow$ current dual solution of restricted master problem (RMP)
3      $\hat{ub} \leftarrow \mathbf{w}^T \boldsymbol{\phi}(\gamma_i, \boldsymbol{\pi})$
4      $y_{il} \leftarrow$ solve pricing problem with upper bound $\hat{ub}$
5      if pricing problem is infeasible then
6          $y_{il} \leftarrow$ solve pricing problem with upper bound $\gamma_i$
7      end
8      add column $y_{il}$ into the RMP
9      add new columns $r_i^{\pm}$ into the prediction model
10     add constraint in form of (5.3)
11     $\mathbf{w} \leftarrow$ solution of (5.2)
12 while column with negative reduced cost exists;
```

dual solution of the master problem. One has to keep in mind the trade off between model complexity (therefore its predictive power) and the saturation of its parameters. This is even enhanced by the online setting of the learning. Therefore, the complexity of the model needs to be as small as possible such that it still provides some reasonable predictions. In our experience, model predicts good values for $n = 3$ number of features. One of them is default bound γ and the others are expected values of $\boldsymbol{\pi}^T\mathbf{x}$ for disjunctive parts of the individual schedule. Although we use uniform distribution over shifts assignments, one can improve this estimate by the estimation of probabilistic distribution (e.g. by Markov assumption) of shifts from columns, that are generated for given employee.

If the pricing problem with predicted bound is not able to find any column, it cannot guarantee that column with negative reduced cost does not exists. In that case, one has essentially two options. When an optimal solution of the pricing problem is not needed (i.e. heuristic stage of column generation) it can proceeds with different subproblem (employee). However, when the proof of non-existence of such column is needed (i.e. before branching for obtaining lower bound on the integer master objective), it is needed to run pricing solver with default bound, i.e. γ. Fortunately, it does not need to start completely from the scratch — partial solutions that were pruned in previous stage due to tighter upper bound can be stored and re-expanded in this stage.

As it can be seen in (5.2), the criterion is the sum of piecewise linear functions. It is worth it to note, that there exists a implementation of Simplex method, that is able to handle this form of the objective natively. It brings significant speedup which can be seen for example with *Gurobi* IP/LP solver, which supports this feature since the version 6.0.

5.2.2 Relation to cover cuts

There is an interesting connection between imposing cutting planes into the primal model and tighten up upper bound on the pricing problem. More precisely, tightening of upper bound has similar effect as adding the *cover cuts* for individual schedules in primal model. Assume that one is able to prove, that in some optimal solution of restricted master problem is not possible to have some individual schedules selected at the same time. This could happen for example if we have upper bound on the whole problem (e.g. from initial heuristic solution) which is lower than a penalty associated with violation of coverage constraints due to a set of patterns $\mathcal{S}$ in the optimum, i.e. when the set $\mathcal{S}$ contains patterns that assign employees to a shift on the same day causing overstaffing for that day.

We impose a constraint which forbids to select this group of individual schedules. Consider adding cover cut in the following form into the primal problem

$$\sum_{(i,l)\in\mathcal{S}} y_{il} \leq |\mathcal{S}| - 1 \tag{5.6}$$

which puts a new variable $\xi \leq 0$ into the dual problem. In order to find a column with negative reduced cost, one has to solve modified pricing problem

$$\gamma_i + \xi > c_{il} - \boldsymbol{\pi}^T \mathbf{x} \tag{5.7}$$

thus, the new upper bound in pricing problem is tighter.

5.2.3 Impact to performance

For the branch and bound based algorithm (i.e. informed state space pricing, see Chapter 3.3.3) we measured how many nodes are expanded during the search. For MlP pricing problem (see Chapter 3.3.2) we computed ratio of runtimes. Results are summarized in Table 5.1. The numbers in the second column indicate the ratio of visited nodes with to the visited nodes without applying our approach in the informed state space pricing. Similarly, the third column represents the ratio of the runtime with to the runtime without applying our approach in the MIP pricing solver.

In Figures 5.3 and 5.4 one can see example run of the proposed algorithm. It can be seen, that from the beginning, parameters of the predictive model are not properly estimated. However, after a few more iterations, the algorithm starts to track bound reasonably well.

Instance	visited nodes ratio [-]	runtime ratio [-]
Millar-2Shift-DATA1	0.95	-
WHPP	0.75	-
Valouxis-1	-	0.79
Azaiez	-	0.55
SINTEF	-	0.94

Table 5.1: Impact of upper bound tightening to the performance. Values < 1 indicates a positive speed up.

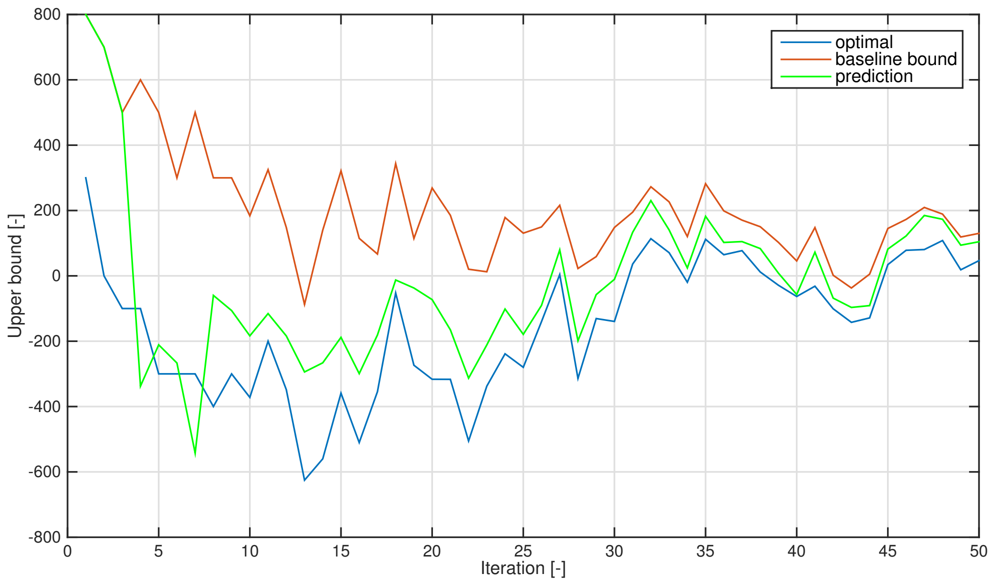


Figure 5.3: Upper bound prediction for subproblems in Millar instance.

5.2.4 Further improvements

The main disadvantage of proposed method is that it uses a hyper parameter ϵ which needs to be chosen. Well-known methods for its tuning like cross-validation are not easily applicable to the online setting of learning. Therefore, it is desired to omit it. In future research, we would like to incorporate it among the other variables which are optimized during training.

Moreover, we would like to investigate the effect of different objective function. In paragraphs above we proposed linear penalty with larger coefficient for under estimating the bound. However, it might be useful to penalize predictions below target value using *step function*, since this is more closer to our desired criterion. Currently, we minimizes the upper bound on the error function.

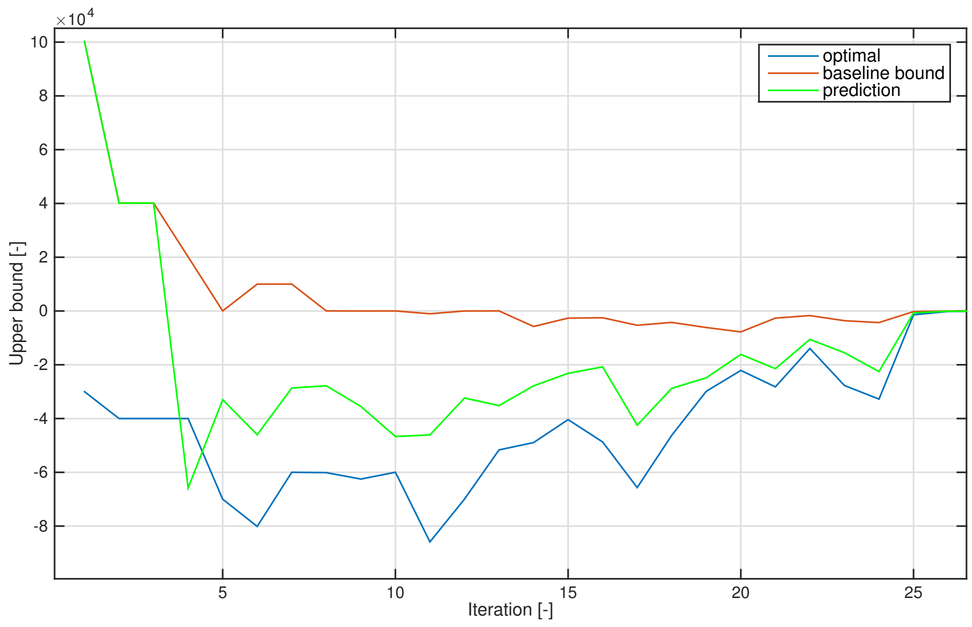


Figure 5.4: Upper bound prediction for subproblems in Azaiez instance.

6 Additional observations

6.1 Motivation

Even though both the column generation and branch and price algorithm have solid mathematical foundations (as described in previous chapters) and it seems straightforward how to implement them, actually, there are some caveats in practice.

Column generation, as it is very elegant in theory, tends to be difficult in practice. It is prone to convergence difficulties, tailing off (Lübbecke and Desrosiers, 2005), dual oscillations (Du Merle et al., 1999), and many other issues. Therefore, in order to have branch and price effective in practice, one has to come up with *a set of tricks* to overcome these problems.

Since there is essentially none published information about the properties of our problem concerning column generation, we will describe important improvements we had to figure out in order to make our algorithm fast. Moreover, in this chapter we will discuss some newly emerged properties of our problem.

6.2 Column generation

6.2.1 Dual variables stabilization

It is commonly known fact that column generation has often problem with the convergence speed (Lübbecke and Desrosiers, 2005). However, in order to improve the master objective, one has to generates high-quality columns (i.e. schedules with low penalization which improves staffing levels). Columns are generated based on the dual prices of the primal constraints, so it may happen that we generate useless columns just because of poor information about an estimate of optimal dual variables.

Known observation is that columns in the optimal solution of the primal problem are usually generated among the last ones (Lübbecke and Desrosiers, 2005), i.e. based on the information close to optimal dual solution $(\boldsymbol{\gamma}^*, \boldsymbol{\pi}^*)$, so it is presumably helpful to not to spend more time than necessary with earlier iterations and we should rather focus to get an estimate of the optimal dual solution quickly. We can see this behavior in Figure 6.1, where the distance between the vector of current dual solution and the optimal one is depicted. For example, at the iteration 21 we are heading away from the optimal solution and just after a few iterations we start coming back. The analogy of this phenomenon is the Newton's method for finding local minima with poorly set step size which overshoots the minima. If we would be able to "smooth out" the sequence of dual solutions, less iterations of column generation would be needed to solve the master problem.

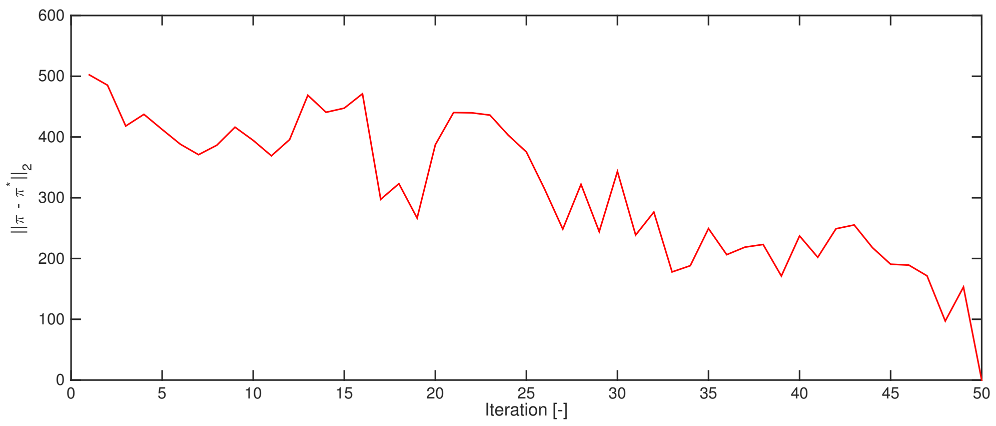


Figure 6.1: Unstabilized column generation in *Millar* instance. Distance from current dual solution to optimal dual solution is not monotonically decreasing. It results in so-called *bang-bang* behavior.

This issue in column generation is commonly addressed by *stabilization methods*. The *trust region method* (Amor and Desrosiers, 2006) solves this problem by imposing new constraints on the dual variables of the master problem. Essentially, it forces values of dual variables to lie within preferred box given by the box constraints (i.e. $lb \leq \pi_{mjk} \leq ub$). This box is re-estimated every time after the new column is added to the primal model. Thus, dual solutions are less likely to oscillate.

However, if the estimation of the *preferred* box is bad, it could even slow down the convergence. Therefore, we should allow dual solution to leave the box for some penalty, if it is needed. This modification is called *trust region method* with ϵ-perturbation (Pigatti et al., 2005). In order to constraint a variable in the dual problem one has to insert new variables (artificial columns) into the primal model. We will show the modification on simplified problem. Consider original (unstabilited) LP and its dual

$$
\begin{aligned}
&\min_{\mathbf{x}} \mathbf{c}^T\mathbf{x} && && \max_{\boldsymbol{\pi}} \mathbf{b}^T\boldsymbol{\pi} \\
\text{subject to} & && \text{subject to} && \\
&\mathbf{A}\mathbf{x} = \mathbf{b} && && \boldsymbol{\pi} \in \mathbb{R}^n \\
&\mathbf{x} \geq \mathbf{0} && && \mathbf{A}^T\boldsymbol{\pi} \leq \mathbf{c}
\end{aligned}
$$

Notice that $\boldsymbol{\pi}$ variables are not constrained. However, putting artificial columns inside stabilized primal problem above yields to

$$\min_{\mathbf{x},\mathbf{z}} \mathbf{c}^T\mathbf{x} - \boldsymbol{\delta}^T(\mathbf{z}^- - \mathbf{z}^+)$$

subject to

$$\begin{aligned} \mathbf{A}\mathbf{x} - \mathbf{z}^- + \mathbf{z}^+ &= \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0} \\ \mathbf{z}^\pm &\leq \boldsymbol{\epsilon} \end{aligned}$$

$$\max_{\boldsymbol{\pi},\boldsymbol{\omega}} \mathbf{b}^T\boldsymbol{\pi} - \boldsymbol{\epsilon}^T(\boldsymbol{\omega}^- + \boldsymbol{\omega}^+)$$

subject to

$$\begin{aligned} \boldsymbol{\delta} - \boldsymbol{\omega}^- \leq \boldsymbol{\pi} &\leq \boldsymbol{\delta} + \boldsymbol{\omega}^+ \\ \mathbf{A}^T\boldsymbol{\pi} &\leq \mathbf{c} \\ \boldsymbol{\omega}^\pm &\geq \mathbf{0} \end{aligned}$$

therefore dual variables are now constrained. They are centered inside the hyperbox given by $[\boldsymbol{\delta} - \boldsymbol{\omega}^-, \boldsymbol{\delta} + \boldsymbol{\omega}^+]$ where $\boldsymbol{\omega}^\pm$ are variables that can be used for its enlargement for $\boldsymbol{\epsilon}$ penalty.

In contrast to the master model described in (Burke and Curtois, 2014), our formulation of it (see Figure 2.3) contains a form of trust region method with ϵ-perturbation by default. Consider constraints (2.39) and (2.40) of the dual master model in Figure 2.4. It can be easily shown, that these essentially restrict dual prices as $-c^o_{mjk} - \phi_{mjk} + \psi_{mjk} \leq \pi_{mjk} \leq c^u_{mjk} - \phi_{mjk} + \psi_{mjk}$. Thus, $\boldsymbol{\pi}$ dual variables are restricted to lie within the hyperbox given by coverage penalties or leave it with the penalty proportional to staffing demands. This key observation was left unnoticed in both works (Maenhout and Vanhoucke, 2010; Burke and Curtois, 2014).

We conducted further experiments with different stabilization method (Du Merle et al., 1999), however, we were not able to obtain any significant improvements. Once again, which is with contrast to the claimed result by (Burke and Curtois, 2014).

6.2.2 Heading-in effect

Column generation also suffers from the effect known as *heading-in* (Lübbecke and Desrosiers, 2005). Column generation procedure needs some feasible solution of restricted master problem to start. In terms of our master model, every employee must have at least one individual schedule assigned. In our case, these columns come from the initial heuristics (see Chapter 3.4).

Column generation then proceeds with the dual information based on these columns and starts generating new ones. The issue is, that this dual information can be arbitrarily bad, thus columns generated based on it have only a little chance to be contained in the optimal solution (Lübbecke and Desrosiers, 2005). Therefore, solving the pricing problem up to the optimality is not practical when being in this stage of column generation.

We addressed this issue by the dynamic control of the gap parameter specifying quality of solutions produced by the pricing algorithm (see Chapter 3.4). Column generation starts with the gap of 30% which is then further narrowed based on the speed of the convergence of the master model. The control equation is given by the simple linear relation

$$gap = c\frac{\mathrm{d}z}{\mathrm{d}t} \approx c\frac{z_{t-1} - z_t}{\Delta} \tag{6.1}$$

where z_t is the objective value of the master problem at the iteration t for some constants c and Δ.

6.2.3 Tailing off effect

Sometimes column generation procedure suffers from a slow convergence — improvement of the objective value of the master model is a little or none during large number of iterations. In that case, it might be useful instead of solving the master model up to the optimality (generating more and more columns) branch earlier and rather try to solve its child nodes optimally.

One has to keep in mind that the objective value of the master model, which is not solved up to the optimality, is not a lower bound on the integer objective. Thus we cannot use it directly for a comparison to the known upper bound in order to prune its child nodes. However, we can still use this suboptimal objective value to get a lower bound using Lagrangian relaxation (see Chapter 4.4).

6.3 MIP pricing

6.3.1 Imposing lower and upper bounds on the objective

One may want to utilize the knowledge he/she has about the objective value of an optimization problem in form of $\min \mathbf{c}^T\mathbf{x}$. If we knew lower and upper bound on the objective we could impose the constraint $lb \leq \mathbf{c}^T\mathbf{x} \leq ub$ in order to prune the search space. However, when a constraint contains all the variables presented in the problem, this turns out to be inefficient (in case of ILOG CPLEX[7] this would even slow down the computation). Imposing the constraint over the objective involving complex linear expression complicates the search space, thus it could make task harder to solve.

More efficient way of incorporating bounds on the objective is to work with nodes in the branch and bound tree constructed by the IP solver. For example, if the LP relaxation in some node yields a value greater than the upper bound given in advance, such node can be cut off. In ILOG CPLEX it can be done by setting parameters *CutLo* and *CutUp* to proper values.

[7]ILOG CPLEX is an LP/IP solver by IBM.

We use those parameters both in MIP pricing solver (see Chapter 3.3.2) and for resolving restricted the master problem as an IP (see Chapter 4.6).

6.3.2 Branching priorities for variables

In every integer programming model we can specify priorities over variables for branching. Simply said, if the continuous relaxation of the model does contain fractional values, the solver will start branch on the variable with the highest priority. If we had some background knowledge, which variables have bigger impact on the objective, it would be desired to settle assignment for them as soon as possible (Gilpin and Sandholm, 2011).

In our MIP model for the pricing problem (see Chapter 3.3.2), variables for assignments during weekends are more important than other ones in most cases (Maenhout and Vanhoucke, 2010). This fact follows from the typical structure of the constraints which restricts frequently shift assignments during Saturdays and Sundays in some way.

6.3.3 Solution pool

Although the main aim of integer programming is usually to find a single optimal solution, MIP-based pricing model described in Chapter 3.3.2 can also provide multiple solutions (columns) at single iteration.

This can be arranged using technique introduced in (Danna et al., 2007) and implemented in ILOG CPLEX. After finding an optimal solution the algorithm enters into the second stage, where it uses information gathered in previous stage such as partial branch and bound tree, integer solutions found so far and other. During this stage, additional solutions are generated, which are not necessarily the optimal ones. These solutions are then stored in a data structure called *solution pool.*

One can specify various parameters of solutions stored in the pool as well as the parameters of the pool itself. For example, we can demand to find additional 5 solutions with the best objective value. Or, as it turns out to be useful for us, store good, *diverse* solutions (defined as the average pairwise Hamming distance on the vectors of integer variables).

In our experiments we found out this feature to be very useful even when there is a little additional time spent for finding another solutions in pool.

6.3.4 Subproblem skipping

As we pointed out in Chapter 3.2, determining whether the column with negative reduced cost exists is $\mathcal{NP}$-complete problem. Moreover, this problem has to be solved for every employee separately, since each of them is a subproblem (recall the block-diagonal matrix $\mathbf{F}$ in Chapter 2.2). When we have a master problem to solve with tens of employees, it becomes difficult even just to check whether the column generation has converged.

However, we can exploit the symmetry of the employees. Consider a subset of employees having the same working contract, skill competency and working preferences. During the solution of the root node in branch and bound tree, the algorithm finds out that for some employee i there is no column with negative reduced cost. The algorithm then switches to the next employee (subproblem) j. However, when $\gamma_i \leq \gamma_j$ holds (i.e. the upper bound on the pricing problem for j is looser than for the i) there is obviously no column with negative reduced cost for employee j too. Thus, we do not have to prove the non-existence of a negative reduced cost column by running the pricing solver. However, this is possible only when no column was added since the last time the subproblem i yielded with no column with negative reduced cost, i.e. dual prices $\boldsymbol{\pi}$ has to be the same.

Similar strategy can be used also in non-root node. However, additional condition for skipping a subproblem has to be considered, since branching constraints were imposed. One has to check whether the set of branching constraints for the subproblem (employee) to be skipped is a super set of the branching constraints for the subproblem which has provably failed (i.e. no column with negative reduced cost exists).

Essentially, we can skip the subproblem whether its relaxation (less constrained subproblem) failed, thus sometimes, even employees with different working contract, workload and preferences can be skipped.

6.4 Symmetry breaking

Symmetries in the problem are often a major difficulty when solving combinatorial optimization tasks (Crawford et al., 1996). By the symmetry we refer a property of a problem (or mathematical model of it) which for the given solution allows us to create a new solution trivially for example just by swapping assignments for distinguishable objects. To be more specific, consider famous *n-queens problem*[8]. If we find some feasible assignment of the queens to positions on the chessboard, we can trivially swap two queens to get a *different* solution.

[8]It is the problem where we are ask to place n queens on a $n \times n$ chessboard in such way, that they do not threaten each other.

Naturally, we do not really consider this as a different solution because queens are not distinguishable to us. We do not perceive them individually. However, for a combinatorial algorithm it is not the case, since the different assignment forms a different state in the state space.

Existence of symmetries for feasible solutions also implies symmetries for non-solutions (infeasible solutions). When we place queens on the chessboard in a way that some of them threat each other, there is a symmetric infeasible solution too. If the algorithm solving the task is not aware of symmetries in the problem it is forced to repeatedly visits these symmetrical infeasible parts of state space before finding the actual solution to the problem. Providing this information to the algorithm can lead to significant improvements (Crawford et al., 1996).

In our problem, symmetries appear in the case we have employees with the same working contract, skill and working preferences. In this case, employees become indistinguishable to us. Symmetry breaking in our task is not as easy as in *n-queens problem*, since it is hard to reason about shift assignments for a nurse in general. However, under some conditions it can be done as we will show in chapter below and it leads to the significant performance gain.

6.4.1 Symmetry breaking by fixing some assignments

We have found a way, how to partially break one of the symmetries in the problem caused by identical employees. To demonstrate it in its simplified form suppose that following three conditions hold

1. employees are identical[9]
2. instance consists only of preferred coverages
3. preferred coverages are not violated in some optimal solution

then setting n equal to a minimum of preferred coverages for all days and shifts and fixing any single shift for any single day for some n distinct employees preserves at least one optimal solution.

The assumption that coverages are not violated in some optimal solution often holds, since high-quality schedules usually do not violate this in order to avoid a large penalty that would incur otherwise. Moreover, we can find out throughout the solving whether our assumption was correct — when we end up with the solution with lower objective than a penalty for violation

[9]In the same sense as stated in Chapter 6.3.4.

	Mo	Tu	We	Th	Fr	**Sa**	**Su**	Mo	Tu	We	Th	Fr	**Sa**	**Su**
Employee A	?	?	E	?	?	?	?	?	?	?	?	?	?	?
Employee B	?	?	?	?	?	?	?	?	?	?	?	?	N	?
Employee C	?	?	E	?	?	?	?	?	?	?	?	?	?	?
Employee D	?	?	?	?	?	?	?	?	?	?	?	?	?	?
⋮														
Employee Z	?	?	?	?	?	?	?	?	?	?	?	?	?	?
required E	5	5	5	5	5	4	4	5	5	5	5	5	4	4
required D	4	4	4	4	4	$\boxed{3}$	3	4	4	4	4	4	3	3
required N	4	4	4	4	4	3	3	4	4	4	4	4	3	3

Figure 6.2: An illustrative example of symmetry breaking. It fixes single assignment for 3 different employees.

of a coverage constraint, then it is clear that at least one optimal solution is contained in the set of solutions without violation of coverage constraints.

This technique can be further extended into the form, where we fix more assignments than stated above. This can be done by fixing days in a more informed way. For example, one could fix all assignments for the first day of the schedule (once again, under the assumption that coverage constraints are not violated). However, we observed that this is not the best way of doing that. Even though we fix more assignments (and thus we are reducing the number of free variables), we still preserve symmetry between employees which has the same shift assigned. Based on our observation it seems, that superior strategy is to fix less shifts, but rather in diverse way — different days, different shifts. In our opinion, the strategy mentioned in paragraph above improves the propagation of dual prices for coverage constraints (in horizontal direction) and, thus, accelerates column generation convergence.

7 Experimental results

In this chapter we will discuss experimental results. In Chapter 7.1, the standard real-world benchmark instances will be described. Chapter 7.1.1 provides results of generic branch and price algorithm under various pricing techniques. We show overall results achieved by our algorithm in Chapter 7.1.2 and, moreover, we will carry out the comparison of impact of speedup techniques.

Next, in Chapter 7.2 we will show how we have developed a new real-world benchmark instance in cooperation with *Motol hospital* in Prague. We have found an optimal schedule for the surgery department and proved its optimality. Moreover, the comparison with the provided hand-made roster shows a significant improvement in roster's objective.

At last, in Chapter 7.3 we discuss obtained results and questions raised regarding to the comparison to other works.

7.1 Real-world benchmarks

We ran our algorithm on benchmark instances taken from (Curtois, 2014) (see Table 7.1). Reported runtimes are measured on a system with 1.3 GHz Intel Core i5 (laptop class) processor with 8 GB of RAM. The algorithm is implemented in Java 7. Maximum runtime was set to 120 s.

Instance	Employees	Shifts	Skills	Days	Constraints
Ozkarahan	14	2	2	7	46
Musa	11	1	3	14	34
LLR	27	3	1	7	216
Millar-2Shift-DATA1	8	3	1	14	88
Millar-2Shift-DATA1.1	8	3	1	14	72
Azaiez	13	3	3	28	143
WHPP	30	4	1	14	450
SINTEF	24	5	1	21	284
Valouxis-1	16	4	1	28	224
Motol-1	12	4	1	29	52

Table 7.1: Parameters of real-world benchmark instances.

Moreover, we created a new real-world benchmark instance in cooperation with *Motol hospital* in Prague (see basic parameters in Table 7.1). We restricted maximum runtime to the 30 minutes, which was the requirement given by the hospital.

It is worth it to note that difficulty of an instance do not only depends on the numbers presented in Table 7.1. It is mostly determined by coverage constrains. When these constraints are not so restrictive (i.e. we have much more employees than desired coverages) one can expect to find optimal solution quickly. Moreover, when an instance do not impose coverage constraints across multiple skills, the problem is obviously decoupled. In our experience, the task become usually harder when penalties for violation of coverage constraints are comparable to penalties of soft constraints in individual schedules.

7.1.1 Generic branch and price

In this section we show results obtained by generic (vanilla) branch and price working according to its definition. Results with two different pricing methods are in Table 7.2. In both cases the *Single-pass* initiation heuristics (see Chapter 4.2.1) was used. For each pricing method, the first column contains objective value of the best solution found within the given time limit. As it can be seen, algorithm performs well only on smaller instances.

Instance	State-space search pricing		MIP pricing	
	ub	runtime [s]	ub	runtime [s]
Ozkarahan	**0**	0.6	**0**	1.1
Musa	**175**	0.6	**175**	0.6
LLR	**301**	24.9	**301**	62.3
Millar-2Shift-DATA1	700	120	**0**	20.7
Millar-2Shift-DATA1.1	700	120	**0**	11
WHPP	–	120	10057	120
Azaiez	–	120	$> 1 \cdot 10^5$	120
SINTEF	–	120	22558	120
Valouxis-1	–	120	20460	120

Table 7.2: Comparison of generic branch and price approach with state-space search pricing solver. Solutions with proven optimality are in bold.

Results suggest that state-space search based pricing solver (see Chapter 3.3.3) is able to outperform MIP pricing (see Chapter 3.3.2) on the smaller instances (up to 14 days planning horizon with 3 shift types). On the other hand, MIP pricing is superior on larger instances.

7.1.2 Overall results

In this section we show results obtained with proposed improvements to the generic branch and price algorithm. In Table 7.3 we report lower bound proven by our algorithm and the best upper bound (solution cost) found. We compared our runtimes to the results published by (Burke and Curtois, 2014). Their results are obtained on hardware comparable to our and includes the sum of runtime of initiation heuristic as well as time spent in branch and price algorithm.

Instance	lb	ub	runtime [s]	(Burke and Curtois, 2014) runtime [s]
Ozkarahan	0	**0**	0.06	5.1
Musa	175	**175**	0.5	5.1
LLR	301	**301**	4.3	5.8
Millar-2Shift-DATA1	0	**0**	0.29	5.1
Millar-2Shift-DATA1.1	0	**0**	2.02	5.1
WHPP	5	**5**	20	22.6
Azaiez	0	**0**	3.3	5.3
SINTEF	0	**0**	61	15.5
Valouxis-1	20	9260	120	914.6
Motol-1	1485	**1485**	991	–

Table 7.3: Computational results on real-world benchmark instances. Solutions with proven optimality are in bold.

In the folllowing sections we will present the performance of the whole proposed algorithm without specified improvement to show its contribution to the final results (presented in Chapter 7.1.2.

7.1.3 Effect of improved MIP pricing

By the improved MIP pricing we refer the pricing method which is able to solve problems up to the specified optimality gap with modified branching priorities for variables (Chapter 6.3). It can be seen it has positive effect on nearly all test instances.

7.1.4 Effect of solution pool

Solution pool is a technique for obtaining multiple solutions for MIP based pricing (see Chapter 6.3.3). Therefore, we used it for all test instances. The size of the pool was set to 15.

Instance	ub	runtime [s]
Ozkarahan	**0**	0.06
Musa	**175**	1.6
LLR	**301**	6.5
Millar-2Shift-DATA1	**0**	0.29
Millar-2Shift-DATA1.1	**0**	2.02
WHPP	**5**	30.2
Azaiez	**0**	4
SINTEF	**0**	62
Valouxis-1	18400	120

Table 7.4: Results achieved without the improved MIP pricing. Solutions with proven optimality are in bold.

Instance	ub	runtime [s]
Ozkarahan	**0**	0.06
Musa	**175**	0.5
LLR	**301**	13.9
Millar-2Shift-DATA1	**0**	0.29
Millar-2Shift-DATA1.1	**0**	2.02
WHPP	8045	120
Azaiez	**0**	70.3
SINTEF	**0**	62
Valouxis-1	17580	120

Table 7.5: Results achieved without the solution pool. Solutions with proven optimality are in bold.

7.1.5 Effect of primal heuristics

Primal heuristics focus on improving upper bounds on integer solution in a short time. Therefore, it helps to speed up even the proof of optimality in cases when the Simplex algorithm finds fractional solution even though there are columns capable of forming optimal integer solution.

7.1.6 Effect of subproblem skipping

In Table 7.7 can be seen that subproblem skipping has overall good influence, especially on *Azaiez* instance. However, we believe that the effect similar to the achieved one by this technique could be done also with some heuristics imposing an order on the subproblems in which they are solved.

Instance	ub	runtime [s]
Ozkarahan	**0**	1.6
Musa	**175**	0.6
LLR	**301**	4.3
Millar-2Shift-DATA1	**0**	5.8
Millar-2Shift-DATA1.1	**0**	6.7
WHPP	**5**	31.2
Azaiez	**0**	60.3
SINTEF	22558	120
Valouxis-1	21420	120

Table 7.6: Results achieved without the primal heuristics. Solutions with proven optimality are in bold.

Instance	ub	runtime [s]
Ozkarahan	**0**	0.06
Musa	**175**	0.5
LLR	**301**	6
Millar-2Shift-DATA1	**0**	0.29
Millar-2Shift-DATA1.1	**0**	2.02
WHPP	**5**	23.7
Azaiez	**0**	46.7
SINTEF	**0**	62
Valouxis-1	18400	120

Table 7.7: Results achieved without the subproblem skipping. Solutions with proven optimality are in bold.

7.1.7 Effect of symmetry breaking

From our test set, the following instances satisfy precondition needed to use the proposed technique of symmetry breaking. Runtimes and solutions achieved without symmetry breaking (Chapter 6.4) are presented in Table 7.8.

Instance	ub	runtime [s]
WHPP	**5**	28
Valouxis-1	11360	120

Table 7.8: Results achieved without the symmetry breaking. Solutions with proven optimality are in bold.

We excluded *Millar* instances from this comparison since they are solved and proved to be optimal due to primal heuristics in the same time even without symmetry breaking. However, when one disables the primal heuristics the speedup about 50% can be observed.

7.2 Motol instance

We teamed up with *Motol hospital* in Prague, Czech Republic, in order to provide to us rules and coverage requirements for one of their departments. It consists of 12 employees which are required to be scheduled on one of three shift types (plus off shift) over 29 days horizon. The rules specify various workloads for individual nurses as well as regulation restrictions for shifts assignment. Moreover, it specifies preferred coverages for each day and shift type. Nurses also have their requests for shift assignments (e.g. *employee H* does not want to have D shift on Thursdays).

	Mo	Tu	We	Th	Fr	**Sa**	**Su**	Mo	Tu	We	Th	Fr	**Sa**	**Su**	Mo	Tu	We	Th	Fr	**Sa**	**Su**	Mo	Tu	We	Th	Fr	**Sa**	**Su**	Mo
Employee A	-	-	E	N	-	N	N	-	-	-	-	N	N	-	-	-	N	N	-	D	N	-	-	-	-	-	-	-	-
Employee B	-	N	N	-	D	N	-	-	N	N	-	-	N	N	-	N	E	N	-	N	N	-	N	-	-	-	-	N	-
Employee C	-	D	D	N	-	-	-	D	E	D	N	-	-	-	-	E	D	-	-	N	-	E	D	N	-	N	-	D	D
Employee E	D	-	-	-	-	-	-	-	-	E	D	-	-	-	D	E	D	-	-	-	-	-	-	E	D	-	-	-	E
Employee F	-	-	-	-	-	-	-	-	-	D	-	-	-	-	-	N	-	-	D	-	D	E	E	E	N	-	N	-	-
Employee G	-	-	E	D	E	D	D	E	D	-	-	-	D	D	-	-	E	E	E	D	-	-	-	D	-	-	-	-	-
Employee H	-	-	D	E	E	-	-	D	E	E	E	D	D	D	-	-	-	-	-	-	-	D	D	E	-	-	D	N	-
Employee I	D	E	E	-	-	-	D	E	-	-	N	-	-	-	E	D	E	D	N	-	-	-	N	-	-	E	D	D	E
Employee J	-	N	-	-	-	D	N	-	N	-	-	E	-	N	-	-	-	-	E	-	D	-	-	-	N	-	N	-	-
Employee K	N	-	E	D	N	-	-	N	-	E	D	-	-	-	D	D	E	D	D	-	-	N	-	E	E	D	-	-	D
Employee L	E	E	-	-	-	-	-	-	-	E	-	E	-	-	E	-	-	E	-	-	-	-	-	E	E	E	-	-	-
Employee M	E	D	E	E	D	-	-	-	D	E	E	D	-	-	N	-	E	-	-	-	-	D	E	D	D	D	-	-	N

Figure 7.1: Optimal solution for *Motol-1* instance. Large consecutive blocks of free days are caused by personal requests (i.e. vacation).

To compare this new instance to previous ones, we would say that nurses in *Motol-1* instance are quite unique (a vast majority of them has own working contract). Moreover, the minimum number of working hours (workload) must be fulfilled otherwise it yields in large penalty. On the other hand, working over permitted workload in contract is not penalized much. Interesting feature is also that a few assignments are frozen — e.g. some employees must be assigned to given shift on given day.

The Motol hospital also provided their solution — in Table 7.9 we compared it to our solution, whose optimality is proven. It can be seen that it significantly improved the objective value and, thus, the overall quality of assignments.

Instance	provided solution	our solution	lower bound[†]
Motol-1	338585	**1485**	1485

Table 7.9: Comparison of our solution to the provided one. It significantly improved the given one. Moreover, it proved that our solution is optimal. [†] lower bound was obtained by the column generation in root node.

We hope that in future we would be able to cooperate with the hospital on regular basis in order to improve quality of rosters for their employees and consequently, the health care quality.

7.3 Discussion

As it can be seen in Table 7.3, our algorithm performed well on majority test instances. It struggled on *SINTEF* instance, which specifies hard coverage constraints. Our master model models it using a large penalty (so-called *Big M*) for understaffing and overstaffing. A vast majority of runtime was spent in column generation, which converged poorly. Therefore, in our opinion, master model was degenerated which caused slow convergence. We would like to address this issue in the future development.

Next, we would like to mention an unclarity of the result obtained by (Burke and Curtois, 2014) for *WHPP* instance. From their published solution can be seen, that they considered some constraints as hard ones, however, this test instance specifies all constraint as soft ones. Therefore, it is not clear under what settings authors did solve the instance. Replacing soft constraints for hard ones would simplify the task. However, we treated constraints as soft ones.

On the *Valouxis-1* instance our algorithm was not able to find an optimal solution in the given runtime. Even though, the found solution with objective value over 9000 may not seem good enough, the converse is true. By examining penalties of constraints in this instance it can be seen that only about 10 soft constraints were violated among all 16 employees over 4 weeks time horizon. Moreover, our algorithm proved that solution with no violation does not exist.

The results by (Burke and Curtois, 2014) show odd feature which was discussed in Chapter 1.3. For example, for the *Valouxis-1* instance, their algorithm finished with suboptimal results after seemingly random time (i.e. no upper limit for runtime). This can be also observed for the results for different benchmark set (Curtois T., 2014) where the same behavior occurred multiple times. This behavior was commented by authors as *"... [algorithm] often finding optimal solution... "*. Therefore, their algorithm is not complete one and, thus, incomparable with ours. Its description is misleading and one could get an impression that it is an exact method although there is no sentence directly claiming that. Furthermore, our algorithm was able to derive stronger lower bound.

We would like to stress once again that one has to keep in mind that comparison with the results by (Burke and Curtois, 2014) is for illustrative purpose only. Their task was simpler since their approach does not guarantee optimality.

In this chapter we presented results achieved by our proposed algorithm. For nearly all test instances, the algorithm was able to find optimal solution and prove its optimality in given time limit. Moreover, to test the suitability of our algorithm for real-world environments, we attempted to find a solution for roster for March 2015 for the surgery department of

Motol hospital in Prague. We were able to find the optimal solution in reasonable time, which improved their original solution's objective over a factor of 220.

8 Conclusion

In this thesis, an exact algorithm for Nurse Rostering Problem was introduced. We studied the generic Branch and Price method in details. The properties of Nurse Rostering Problem were studied as well. We designed a Branch and Price based algorithm for solving Nurse Rostering Problem allowing expressive both coverage constraints and working regulations constrains. Properties of the problem were exploited in order to speed up the algorithm. A number of improvements for generic Branch and Price algorithm was developed also. Those exploited properties of our problem resulted in tractability of proving optimality on medium-sized real-world instances.

Since we have shown that medium-sized instances can be solved up to the optimality in a range of seconds to minutes, the only benefit of metaheuristic approaches (i.e. runtime) for them is discarded. Furthermore, we tested our algorithm on the new real-world benchmark instance, which we created in cooperation with employees of a large hospital in Prague.

Moreover, we have shown that the information derived by machine learning algorithms during the number of iterations of $\mathcal{NP}$-hard problems can be used in order to speed up the optimization algorithm in the future while preserving exactness of the algorithm. This unveils a great opportunity for improving the performance on related problem instances using information taught in the past.

To the best of our knowledge, this work represents the only exact Branch and Price based algorithm for Nurse Rostering Problem applied to real-world instances with soft constraints. Moreover, it was able to outperform current best solution times for public benchmark instances.

8.1 Future work

In this work, we focused on the possibility of gaining information about hard optimization problems by *observing*, i.e. feature-based machine learning. The next work should focus on gaining information by *reasoning*, i.e. deriving logically correct properties of given problem instance which are not contained explicitly in the given problem.

In future research we would like to investigate the question how to automatically derive constraints (i.e. cuts, symmetry breaking constraints, etc.) for the given problem instance. Especially, how to automatically break symmetries in such instances. The approach based on searching for graph isomorphisms described in (Crawford et al., 1996) has been used successfully for derivation of symmetry breaking predicates in *Pigeonhole problem* and *n-queens problem*. The question is, whether it is possible to apply it for integer programming problems.

Moreover, a large number of possible improvements of the Branch and Price algorithm are left unexplored. One of them is designing a of different objective function for the pricing problem, since the selection of solution with the minimal reduced cost does not guarantee the fastest convergence of column generation. The other thing that needs to be addressed is the choice of the algorithm for the master problem. The Simplex algorithm finds solutions of the restricted master problem at some vertex of feasibility polyhedron while e.g. the *interior point method* tends to find solutions in the middle of its face. It was reported (CGC, 2014), that these different dual solutions may have a significant effect on the convergence behavior (intuitively, when the dual solution is placed in the middle of a face, it is more likely that a cutting plane cuts off more space than in the case of the vertex placement). This is indirectly connected with the open question, which order the subproblems should be solved in. In our experience, proper ordering of pricing problems (i.e. problems for individual nurses) has positive influence on the speed of convergence of the column generation. However, in this work, we did not study its effect in details.

Some improvements can be done for the MIP pricing model too. An interesting opportunity is to use *lazy constraints* for checking pattern matches. Typically, only a few of them are active in the optimum, thus all of them do not necessary need to be present in the model.

Another interesting modification of our algorithm is to turn it into a powerful heuristic — e.g. do not branch, round up almost decided assignments, solve pricing problem in more relaxed way, etc. Since we know the lower bound given by column generation, we are able to determine the quality of such suboptimal solutions. This is a great advantage over various metaheuristics, which are not able to establish how far their solution is from the optimal one. With this knowledge, one can drive the decision, whether the additional solution time is likely to improve the objective in a considerable way.

We believe, that all those further improvements and other unmentioned ones will lead to the computational tractability of proving optimality of solutions on the largest instances.

9 Bibliography

Uwe Aickelin and Kathryn Dowsland. Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. *Journal of Scheduling*, 3(3):139–153, 2000.

Uwe Aickelin and Kathryn A Dowsland. An indirect genetic algorithm for a nurse-scheduling problem. *Computers & Operations Research*, 31(5):761–778, 2004.

Farid Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimization*, 5(1):13–51, 1995.

Hatem Ben Amor and Jacques Desrosiers. A proximal trust-region algorithm for column generation stabilization. *Computers & Operations Research*, 33(4):910–927, 2006.

Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, Martin WP Savelsbergh, and Pamela H Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329, 1998.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Edmund Burke, Patrick De Causmaecker, and Greet Vanden Berghe. A hybrid tabu search algorithm for the nurse rostering problem. In *Simulated evolution and learning*, pages 187–194. Springer, 1999.

Edmund Burke, Peter Cowling, Patrick De Causmaecker, and Greet Vanden Berghe. A memetic approach to the nurse rostering problem. *Applied intelligence*, 15(3):199–214, 2001.

Edmund K Burke and Tim Curtois. New approaches to nurse rostering benchmark instances. *European Journal of Operational Research*, 237(1):71–81, 2014.

Edmund K Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The state of the art of nurse rostering. *Journal of scheduling*, 7(6):441–499, 2004.

Edmund K Burke, Jingpeng Li, and Rong Qu. A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems. *European Journal of Operational Research*, 203(2):484–493, 2010.

Yair Censor. Pareto optimality in multiobjective problems. *Applied Mathematics and Optimization*, 4(1):41–59, 1977.

CGC. Acceleration strategies and column generation heuristics. *Summer School on Column Generation*, 2014.

Brenda Cheang, Haibing Li, Andrew Lim, and Brian Rodrigues. Nurse rostering problems—-a bibliographic survey. *European Journal of Operational Research*, 151(3):447–460, 2003.

James Crawford, Matthew Ginsberg, Eugene Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. *KR*, 96:148–159, 1996.

T. Curtois. Employee shift scheduling benchmark data sets. `http://www.cs.nott.ac.uk/~tec/NRP/`, 2014. Accessed: 2014-10-14.

Qu R. Curtois T. Computational results on new staff scheduling benchmark instances. Technical report, ASAP Research Group, School of Computer Science, University of Nottingham, 2014.

Emilie Danna, Mary Fenelon, Zonghao Gu, and Roland Wunderling. Generating multiple solutions for mixed integer programming problems. In *Integer Programming and Combinatorial Optimization*, pages 280–294. Springer, 2007.

Jacques Desrosiers and Marco E Lübbecke. *A primer in column generation*. Springer, 2005.

Olivier Du Merle, Daniel Villeneuve, Jacques Desrosiers, and Pierre Hansen. Stabilized column generation. *Discrete Mathematics*, 194(1):229–237, 1999.

Andrew Gilpin and Tuomas Sandholm. Information-theoretic approaches to branching in search. *Discrete Optimization*, 8(2):147–159, 2011.

Atsuko Ikegami and Akira Niwa. A subproblem-centric model and approach to the nurse scheduling problem. *Mathematical Programming*, 97(3):517–541, 2003.

Stefan Irnich and Guy Desaulniers. *Shortest path problems with resource constraints*. Springer, 2005.

Andrzej Jaszkiewicz. A metaheuristic approach to multiple objective nurse scheduling. *Foundations of Computing and Decision Sciences*, 22(3):169–184, 1997.

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

Bernhard Korte, Jens Vygen, B Korte, and J Vygen. *Combinatorial optimization*. Springer, 2002.

Eugene L Lawler and David E Wood. Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719, 1966.

Marco E Lübbecke and Jacques Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

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

Julien Menana and Sophie Demassey. Sequencing and counting with the multicost-regular constraint. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 178–192. Springer, 2009.

Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

Kevin P Murphy. *Machine learning: a probabilistic perspective.* MIT press, 2012.

Panos M Pardalos and Jue Xue. The maximum clique problem. *Journal of global Optimization*, 4(3):301–328, 1994.

Sanja Petrovic, Gareth Beddoe, and Greet Vanden Berghe. Storing and adapting repair experiences in employee rostering. In *Practice and Theory of Automated Timetabling IV*, pages 148–165. Springer, 2003.

Alexandre Pigatti, Marcus Poggi De Aragão, and Eduardo Uchoa. Stabilized branch-and-cut-and-price for the generalized assignment problem. *Electronic Notes in Discrete Mathematics*, 19:389–395, 2005.

Stuart Russell, Peter Norvig, and Artificial Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25, 1995.

Haroldo G Santos, Túlio AM Toffolo, Rafael AM Gomes, and Sabir Ribas. Integer programming techniques for the nurse rostering problem. *Annals of Operations Research*, pages 1–27, 2014.

Barabara M Smith, Kostas Stergiou, and Toby Walsh. Modelling the golomb ruler problem. *Research report series — University of Leeds School of Computers Studies LP SCS RR*, 1999.

James M Tien and Angelica Kamiyama. On manpower scheduling algorithms. *Siam Review*, 24(3):275–287, 1982.

Christos Valouxis and Efthymios Housos. Hybrid optimization techniques for the workshift and rest assignment of nursing personnel. *Artificial Intelligence in Medicine*, 20(2):155–175, 2000.

Marjan van den Akker, Han Hoogeveen, and Steefvan de Velde. Combining column generation and lagrangean relaxation to solve a single-machine common due date problem. *INFORMS Journal on Computing*, 14(1):37–51, 2002.

D Michael Warner. Scheduling nursing personnel according to nursing preference: A mathematical programming approach. *Operations Research*, 24(5):842–856, 1976.

D Michael Warner and Juan Prawda. A mathematical programming model for scheduling nursing personnel in a hospital. *Management Science*, 19(4-part-1):411–422, 1972.

Christopher KI Williams and Carl Edward Rasmussen. Gaussian processes for machine learning. *the MIT Press*, 2(3):4, 2006.

Appendix A CD Content

In Table A.1 are listed names of all root directories on CD.

Directory name	**Description**
thesis	Master's thesis in pdf format.
thesis_sources	latex source codes
code	source code of implemented algorithm

Table A.1: CD Content