CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

MASTER'S THESIS

Alternative Pricing in Column Generation for Airline Crew Rostering

EMILY CURRY

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

Thesis for the Degree of Master of Science

Alternative Pricing in Column Generation for Airline Crew Rostering

Emily Curry

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology and University of Gothenburg
SE – 412 96 Gothenburg, Sweden
Gothenburg, August 2018

Alternative Pricing in Column Generation for Airline Crew Rostering
EMILY CURRY

Supervisor: Ann-Brith Strömberg, Department of Mathematical Sciences
Advisor: Alexander Kruuse, Jeppesen
Examiner: Ann-Brith Strömberg, Department of Mathematical Sciences

Master's Thesis 2018
Department of Mathematical Sciences
Chalmers University of Technology
University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2018

Alternative Pricing in Column Generation for Airline Crew Rostering
EMILY CURRY
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

In airline crew rostering, the objective is to create personalized schedules, i.e., rosters, for a set of crew members. Because of the large number of possible rosters that could be formed, the problem is solved using column generation, where each column corresponds to a specific roster. The pricing problem, which is the problem studied in this thesis, is then defined as to find legal rosters with the potential of improving the current solution. Since the rules and regulations regarding rosters vary between airlines, we have chosen to treat the pricing problem as a black-box optimization problem.

Three different methods for solving the black-box pricing problem have been implemented. The first method uses binary particle swarm optimization (BPSO) to search for new rosters. The other two methods use surrogate modeling to fit a non-linear surrogate function to a set of sampled rosters using radial basis functions. The surrogate function was then either linearly approximated, so that a shortest path problem could be set up and solved, or solved heuristically by a BPSO method.

The three methods have been evaluated on five real-world test cases. For each test case, a large number of different pricing problems are solved. Our comparison of the methods' performance shows that the method using BPSO performed the best, followed by the surrogate modeling approach without the linear approximation.

Keywords: airline crew rostering, column generation, pricing problem, black-box optimization, binary particle swarm optimization, surrogate modeling, shortest path.

Acknowledgements

First of all I would like to thank my supervisor at Jeppesen, Alexander Kruuse, for the help and support, and for always taking the time to discuss new ideas.

I also want to thank the Jeppesen office in Gothenburg, and especially Kristoffer Ptasinski and Joakim Karlsson, for giving me the opportunity to work with such an interesting problem in such an inspiring environment. I want to thank Andreas Westerlund for initially introducing me to the airline crew rostering problem and my Jeppesen roommate, Elin Blomgren.

Thanks to Ingrid Johansson at the Chalmers Library for helping me find relevant literature in the beginning of the thesis work.

Finally, special thanks to my supervisor and examiner at Chalmers, Ann-Brith Strömberg, for the useful feedback and guidance regarding the research and writing of this thesis.

Emily Curry, Gothenburg, May 2018

Glossary

activities See description for *tasks*.

activity constraints The constraints which ensure that each task is covered by the correct number of crew members.

airline crew rostering problem The problem of assigning a set of tasks to a set of airline crew members such that all tasks are covered and all crew members are assigned to a roster.

assignment constraints The constraints which specify that each crew member has been assigned to exactly one roster.

binary particle swarm optimization (BPSO) Population-based optimization technique for binary optimization problems inspired by swarming behavior in nature.

black-box optimization Solving an optimization problem where the objective function has an unknown algebraic structure.

branch-and-price A combination of column generation and branch-and-bound to solve large integer or mixed integer constrained programs.

cognitive component A constant used to determine the influence of a particle's best known position when updating the particle's velocity in BPSO.

column generation An algorithm for solving large linear programs in which the problem is defined over a sufficiently large subset of variables and new variables are added in an iterative process.

crew complement The number of crew members that a pairing should be assigned to and which ranks these crew members should have.

crew pairing problem Forming sequences of flight legs into anonymous pairings that start and end in the same base such that the crew need on each flight is fulfilled by all pairings.

horizontal rules The rules that concern single rosters, for example most rules regarding crew rostering.

inertia weight A parameter used to handle the trade-off between exploration and exploitation when updating a particle's velocity in BPSO.

particle swarm optimization (PSO) Population-based optimization technique for continuous optimization problems inspired by swarming behavior in nature.

qualitative rules The rules that affect the quality of the solution, for example solution robustness.

radial basis functions A function whose value at each point depends only on the distance between that point and the origin.

Rave The rule modeling language for rules and regulations for crew and operations products developed at Jeppesen.

roster A sequence of tasks assigned to a specific crew member.

social component A constant used to determine the influence of the globally best position when updating a particle's velocity in BPSO.

surrogate model A analytic model that approximates the behavior of a complex system.

tasks Activities such as pairings, training activities, ground duties and reserves, that should be assigned to the crew members.

vertical rules The rules that regard several rosters, for example the number of crew members a pairing should be assigned to.

Acronyms

BPSO Binary particle swarm optimization.

IMP Integer master problem.

MP Master problem.

PSO Particle swarm optimization.

RBF Radial basis function.

RMP Restricted master problem.

Contents

1
Introduction

The airline industry employs millions of people and has an annual turnover of billions of dollars. In 2018, the number of employees in the aviation industry is expected to rise over 2.7 million people and the profit to hit $38.4 billion [1]. Within an industry of this size, the need for efficient optimization solutions is high. Optimization problems range from long-term planning to solutions concerning the day of operations. Efficient solutions can result in significant cost savings for an airline and better working conditions for the crew, but because of the large operational scale, the complex rules, and different cost aspects, the resulting optimization problems are very time consuming to solve.

Jeppesen is a global company that specializes in providing solutions to make airlines more efficient and safe, and their products are used by major airlines across the world. In the Swedish office in Gothenburg the main focus area is software for Crew and Operations management. One of the products developed at the Gothenburg office regards crew rostering, which is an important part of airline operations. A *roster* is a personalized schedule, i.e. a list of activities which should be assigned to a certain crew member. In the *airline crew rostering problem*, the aim is to assign a set of tasks to a set of crew members such that all tasks are covered and each crew member has been assigned to a roster. This master's thesis has been conducted together with Jeppesen, and investigate alternative methods for generating legal rosters that are independent of the rule set regarding the rosters.

1.1 The airline crew rostering problem

Crew planning is usually divided into two phases: crew pairing and crew rostering. In the *crew pairing problem* sequences of flight legs are formed into anonymous pairings such that the crew need on each flight is fulfilled. Each pairing has specifications regarding the number of crew members that the pairing should be assigned to and which positions (ranks) these crew members must have. These specifications are referred to as *crew complement.*

In crew rostering, which this thesis concerns, personalized rosters are created for individual crew members. Each roster will consist of pairings that are scheduled together with other tasks, such as ground duties, reserve duties and off-duty blocks. The aim is to assign all tasks such that all pairings are covered by the correct number of crew members with the correct rank and such that each crew member has been assigned to a roster. Rostering is usually done for a period of one month at a time and there can be thousands of crew members and pairings that should be scheduled.

The crew rostering problem is interesting both from the airline's point of view as well as from a crew perspective. Crew costs are the second largest expense of an airline, which means that significant cost savings can be made by creating efficient schedules for the crew. However, also individual crew preferences as well as favorable roster attributes, such as fairness aspects, can be combined with the cost efficiency objective. Therefore, solving the rostering problem can result in more fair schedules and better working conditions, which in turn will increase the satisfaction of the crew.

As described by Kohl and Karisch [2], there are typically four kinds of objectives present in rostering. The first is related to monetary cost. The second is an objective that will result in a robust solution, i.e., to form rosters that are less sensitive towards disruptions that might occur when used in practice. The third objective is to favor particular roster attributes, for example equal assignment between the crew, and the fourth is related to individual preferences.

As can be expected, there is a large number of complex rules that concern rosters, for example working rules and regulations, vacation periods, and training requirements. The rules often depend on which crew member they concern and vary between airlines. Kohl and Karisch distinguish between three different types of rules, namely *horizontal rules*, *vertical rules*, and *qualitative rules.* The horizontal rules cover most of the rules regarding crew rostering and concerns single rosters. Examples of such rules concern rest times and crew qualifications. The vertical rules regard several rosters and can for example be the number of crew members that a pairing should be assigned to. The last type, the qualitative rules, affect the quality of the solution by, for example, ensuring robustness and omitting unattractive solutions. These are often introduced by the airlines themselves and are not due to legislation or contractual agreements.

Because of the scale of the crew rostering problem and the large number of rules, efficient solution methods have to be applied in order to find a good solution.

1.2 Background and aim

The number of rosters that could be formed from the set of pairings is for practical instances too large to be expressed explicitly as well as stored on a computer. Therefore, the crew rostering problem is solved using *column generation* [3] where columns corresponding to personalized rosters are iteratively generated. The original problem is first continuously relaxed and then divided into a *master problem* and a subproblem called the *pricing problem*. The master problem when defined on a subset of all possible rosters is called the *restricted master problem*. The optimal set of rosters is found in an iterative process by alternating solving the restricted master problem and the pricing problem.

In the pricing problem, new rosters are generated and then added to the restricted master problem. The pricing problem is solved for one crew member at a time, and hence several different and separate pricing problems are typically solved in each column generation iteration. Since the quality of the solution highly depends on which rosters are generated, the pricing problem is an important step of the column generation algorithm. The objective in the pricing problem is to generate rosters that are cheap to assign to and that are likely to improve the solution to the restricted master problem.

The rosters found in the pricing problem must be legal in the sense that they fulfill all crew specific rules and no illegal rosters are allowed to enter the restricted master problem. Since the rules and regulations concerning the crew differ between airlines, Jeppesen has chosen to separate the core algorithm, i.e., the column generation algorithm, from the implementation of the rules and use pricing methods that are independent of the rules and regulations. This offers some difficulties when solving the problem since a roster is only evaluated using a simple yes/no question regarding its legality, and by its corresponding cost. Because of this black-box interface, the structure of the subproblem must be considered unknown; together with the fact that the rules and costs are often nonadditive, the pricing problem in itself is a challenging optimization problem.

The focus of this thesis is the crew rostering problem, and more specifically solving the pricing problem in column generation for airline crew rostering. Because of the mentioned properties, heuristics are used to solve the pricing problem and therefore, it is always interesting to investigate alternative methods to found out if the performance could be further enhanced. The aim is to find, implement, and compare alternative methods for solving the pricing problem. The new methods should be evaluated within the existing column generation framework using real-world data provided by Jeppesen.

1.3 Limitations

This thesis will be limited to only investigate alternative methods to solve the pricing problem in the existing column generation framework for the crew rostering problem at Jeppesen. This means that methods used for solving the restricted master problem or finding an integer solution will not be discussed.

The methods implemented in this thesis will be compared with each other. Since the current pricing methods used at Jeppesen are proprietary, they will not be included in the comparison. Instead, the aim of this thesis is to investigate if the alternative pricing methods are able to solve the pricing problems for different problem instances and which method that has the best performance.

In order for the methods to be evaluated within the column generation framework, some consideration to computational time must be taken into account and therefore, for computational reasons, some solution methods might not be appropriate to implement. Apart from that, no specific run time limitations apply.

The methods should work for any given problem instance, i.e. it should be possible to apply the methods using data (including rules) independent of which airline the data comes from. However, the performance will only be evaluated using a few chosen problem instances supplied by Jeppesen.

1.4 Related work

The crew rostering problem in the airline industry has been studied in several journal articles [4, 5, 6, 7]. Similar rostering problems can also be found in connection to for example, the railway industry [8, 9], and hospitals [10].

Because of the complexity of the rostering problem, several metaheuristics have been developed and applied. Lučic and Teodorovic [4] propose a simulated annealing method to solve small to medium size problem instances. First a heuristic is used to create an initial feasible solution which is then improved using simulated annealing. A similar approach is presented by Hadianti et al. [5] for solving the rostering problem at the airline carrier Garuda Indonesia. Examples of other metaheuristics that have been used are scatter search [11] and genetic algorithms [12].

One of the most common approaches to solve the crew rostering problem is to use a column generation technique, where the subproblem of generating new rosters is formulated as a constrained shortest path problem. In Gamache et al. [6], column generation is used to solve the rostering problem for airline crew; a generalized set partitioning model [13] forms the master problem and the pricing problem is formulated as a shortest path problem with resource constraints [14]. The shortest path network is constructed for each crew member, the arcs representing all pairings for which the crew member is qualified and which do not coincide with pre-assigned activities. Additional constraints that could not be modeled within the network are represented using resource variables. Similar approaches are also presented by Fahle et al. [15], Sellmann et al. [16] and Gamache and Soumis [7].

Since the cost of a roster might not be additive, i.e. the marginal cost of adding a pairing to a roster might not equal the cost of adding the same pairing to another roster, solving the shortest path problem while assuming additive costs is a heuristic approach to solve the pricing problem. Attempts to solve nonadditive shortest path problems are presented by for example Gabriel and Bernstein [17] and Chen and Nie [18], both relying on a specific form of the objective function which limits the use to only certain nonadditive shortest path problems.

The column generation framework for the rostering problem at Jeppesen stands out in comparison with the previously mentioned articles because of its black-box interface for the crew-specific rules and constraints; see Section 3.2. In most published articles on this topic the constraints are directly available to the optimizer, which simplifies the construction of the constrained shortest path problem.

Column generation together with black-box constraints is found in Massen et al. [19] where a solution method for the vehicle routing problem with some unknown constraints is presented. To generate new routes, a ”pheromone”-based heuristic is proposed where a set of ”ants” are guided by the current relaxed solution to build new potential routes. However, the black-box only regards the feasibility of a route and not its cost.

One way of looking at the pricing problem is to strictly consider it as a black-box optimization problem, and apply solution methods developed for these kinds of problems. *Black-box optimization* refers to optimizing an objective function with an unknown algebraic structure, i.e. no gradient information can be obtained. A review of derivative-free optimization methods are given by Conn et al. [20].

For some black-box optimization problems, different evolutionary algorithms or swarm-based methods such as particle swarm optimization [21] can be used. However, these types of methods typically require many function evaluations. Another common approach is to create a surrogate which approximates the black-box function and which can be used to search for an optimal solution. Nakayama et al. [22] present a method using support vector machines for optimizing black-box functions and Björkman and Holmström [23] present a method that uses radial basis functions.

1.5 Thesis outline

The remainder of this thesis is structured as follows. Before a proper problem formulation of the airline crew rostering problem and the pricing problem of generating rosters can be presented, an introduction to column generation must be given. In Chapter 2 column generation as a method to solve large integer linear optimization problems is presented and in Chapter 3 a detailed description is given of how the rostering problem can be modeled within the column generation framework. In this chapter, a further description of the pricing problem in column generation for crew rostering and the difficulties that arise when solving the problem are presented. Then, in Chapter 4 and 5 the alternative solution methods used to solve the pricing problem are described. The computational results are presented in Chapter 6 and conclusions in Chapter 7.

2

Column generation for large linear programs

Column generation is an algorithm for solving large linear programs where the number of variables is too large to be considered explicitly. An example of such a problem is the crew rostering problem where the number of potential rosters that could be assigned to each crew member quickly becomes very large. The number of rosters grows exponentially with the number of assignable tasks, which means that for a problem with, say 20 tasks, the number of rosters (legal and illegal) is in the order of $2^{20} = 1\,048\,576$.

Therefore, in this chapter, the column generation scheme for solving large linear and integer programs is presented such that a description of how the crew rostering problem is modeled can be presented in Chapter 3.

2.1 The column generation scheme

Consider a general linear program with m constraints and n decision variables of the form

$$\begin{aligned} \min \ & \boldsymbol{c}^\top \boldsymbol{x}, \\ \text{s.t.} \ & A\boldsymbol{x} = \boldsymbol{b}, \\ & \boldsymbol{x} \geq \boldsymbol{0}, \end{aligned} \tag{2.1}$$

where n is very large and $\boldsymbol{c} \in \mathbb{R}^n$, $\boldsymbol{x} \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $\boldsymbol{b} \in \mathbb{R}^m$. This problem is called the master problem (MP) and can be solved using, for example, the simplex method [24, Ch. 1,3]. Let $\boldsymbol{x}_B \geq 0$ be an initial basic feasible solution with associated basis matrix B and cost coefficients $\boldsymbol{c}_B$. This means that B is an $m \times m$ nonsingular matrix formed from m of the columns in A. The $n - m$ variables corresponding to the columns in A that are not in B are called nonbasic variables and the rest of the variables are called the basic variables. The vector $\boldsymbol{x}_B \in \mathbb{R}^m$ contains the basic variables and solves $B\boldsymbol{x}_B = \boldsymbol{b}$. The cost vector $\boldsymbol{c}_B \in \mathbb{R}^m$ consists of the elements in $\boldsymbol{c}$ which correspond to the columns in B. The simplex multipliers associated with the basis satisfy

$$\boldsymbol{\pi}^\top = \boldsymbol{c}_B^\top B^{-1} \tag{2.2}$$

and can always be obtained by the simplex method. In each iteration, a search for a new variable to price out from the nonbasic variables and enter the basis is performed. This is called the pricing step, where the objective is to find

$$s \in \arg\min_{j\in\mathcal{J}}\{\bar{c}_j := c_j - \boldsymbol{\pi}^\top \boldsymbol{a}_j\}, \tag{2.3}$$

where $\mathcal{J}$ is the set of indices of the columns in matrix A. If column s, consisting of the column $\boldsymbol{a}_s$ in matrix A and cost c_s, satisfies $\bar{c}_s < 0$, the current basic solution may be improved by introducing the corresponding variable x_s in the basis. However, if there are very many columns, i.e. $|\mathcal{J}|$ is very large, an explicit search to find $\min_{j\in|} \bar{c}_j$ by computing each $\bar{c}_j$, is impossible. Therefore the problem is defined over a subset $\mathcal{J}' \subseteq \mathcal{J}$. This is called the restricted master problem (RMP), and is formulated as

$$\begin{aligned} \min \; & \sum_{j\in\mathcal{J}'} c_j x_j, \\ \text{s.t.} \; & \sum_{j\in\mathcal{J}'} \boldsymbol{a}_j x_j, = \boldsymbol{b} \\ & \qquad x_j \geq 0, \forall j \in \mathcal{J}'. \end{aligned} \tag{2.4}$$

Starting from a feasible solution to the RMP, let $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{\pi}})$ be the primal and dual optimal solution of the RMP. Assume that the columns $\boldsymbol{a}_j$, $j \in \mathcal{J}$, are implicitly given as elements of a nonempty set $\mathcal{A}$, and that the cost component c_j can be computed from $\boldsymbol{a}_j$. The problem of finding a column to enter the basis is to solve the so-called pricing problem

$$\bar{c}_j^* := \min\{c(\boldsymbol{a}_j) - \bar{\boldsymbol{\pi}}^\top \boldsymbol{a}_j \mid \boldsymbol{a}_j \in \mathcal{A}\}. \tag{2.5}$$

If $\bar{c}_j^* < 0$, the corresponding column $\boldsymbol{a}_j$ should be added to the RMP. Re-optimizing the RMP will then give a new set of dual and primal solutions, which are "better" than the previous ones, and that are used to find a new column to enter the RMP. On the other hand, if $\bar{c}^* \geq 0$, then none of the reduced cost coefficients $\bar{c}_j$ are negative, hence no choice of entering variable to the basis can be made and an optimal solution to the RMP is in fact an optimal solution to the MP as well. When solving the subproblem, typically only a small subset of the columns in $\mathcal{A}$ are examined and these are then generated only when needed, thus the name column generation.

Depending on the structure of $\mathcal{A}$ and the cost-function $c(\cdot)$, different techniques can be used to solve the subproblem. The only requirement for a column to enter the RMP is that it should have negative reduced cost. Therefore, in the pricing step of column generation, it is not necessary to solve the pricing problem to optimality as long as a column with negative reduced cost has been found. Also, in each iteration, several columns with negative reduced cost can be selected to enter the RMP. The set of columns in the next iteration is given by $\mathcal{J}_{s+1} := \bar{\mathcal{J}}_s \cup \mathcal{J}_s$ where $\bar{\mathcal{J}}_s$ is the set of columns with negative reduced cost found when solving the pricing problem and $\mathcal{J}_s$ is the set of columns in the RMP at iteration s.

Assuming that the solution method for solving the pricing problem finds the column with minimal reduced cost, then, when no column with negative reduced cost can be found, the optimal solution to the original problem has been found.

2.2 The branch-and-price method for solving large integer programs

Now, consider the integer minimization problem

$$\begin{aligned} \min \ & \boldsymbol{c}^\top \boldsymbol{x}, \\ \text{s.t.} \ & A\boldsymbol{x} = \boldsymbol{b}, \\ & \boldsymbol{x} \in \{0,1\}^n. \end{aligned} \tag{2.6}$$

This problem is called the *integer master problem* (IMP) and by relaxing the binary constraints the master problem from (2.1) is obtained. To find an integer solution using column generation, the method must be combined with an appropriate integer programming technique such as branch-and-bound [25]. The combined method is called *branch-and-price* and was originally proposed by Barnhart et al. [26].

In branch-and-price, column generation is used to solve the continuous relaxation of each node of a branch-and-bound tree. If the solution given by the column generation algorithm is integer, the optimal solution in the node of the branch-and-bound tree has been found. Otherwise, some branching rule is applied. The complete branch-and-price scheme for finding an optimal solution to an integer optimization problem using column generation is illustrated in Fig. 2.1.

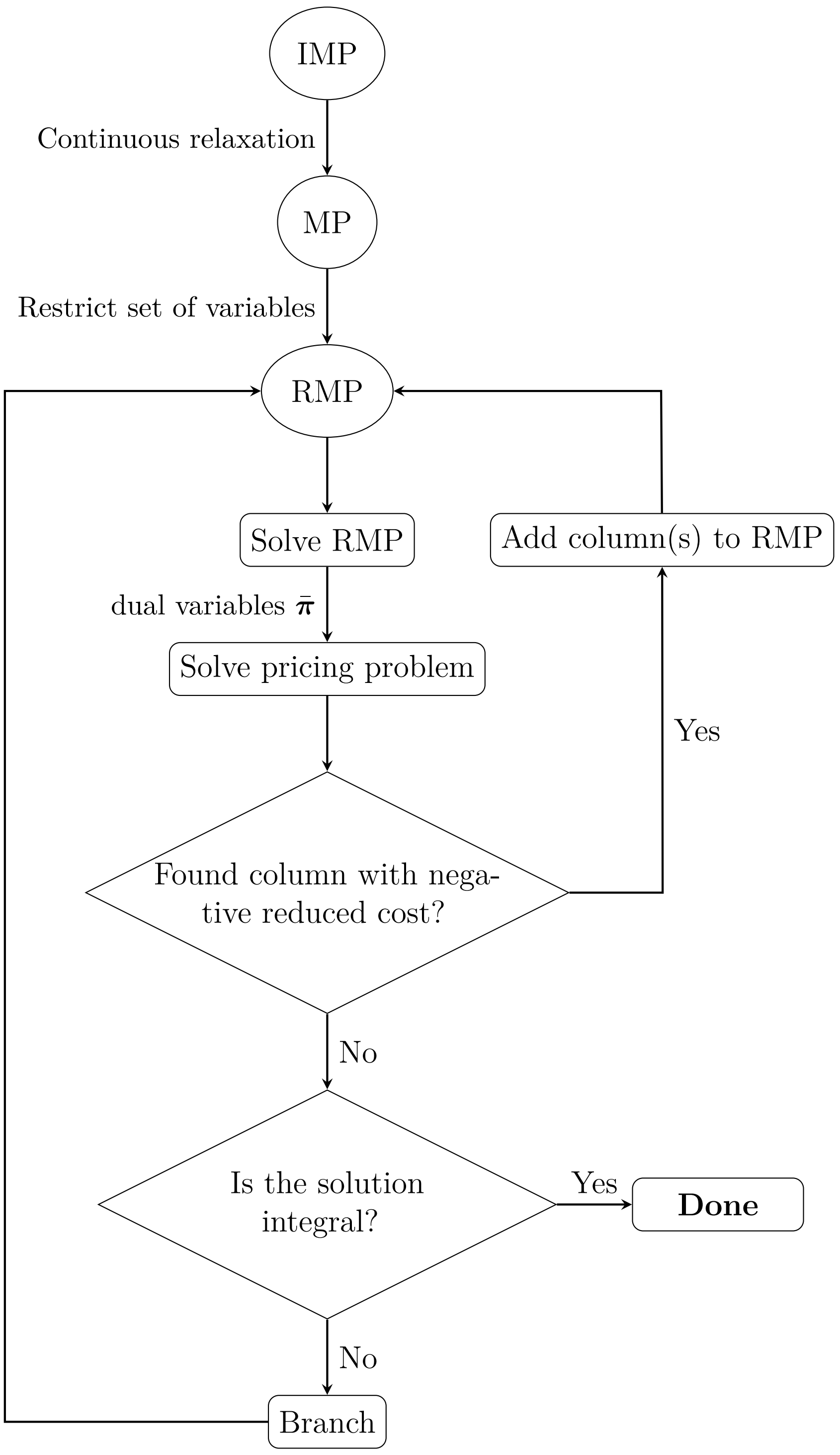

IMP
Continuous relaxation
MP
Restrict set of variables
RMP
Solve RMP
dual variables $\bar{\boldsymbol{\pi}}$
Solve pricing problem
Add column(s) to RMP
Yes
Found column with negative reduced cost?
No
Is the solution integral?
Yes
Done
No
Branch

Figure 2.1: The complete branch-and-price scheme (column generation together with branch-and-bound).

3

Problem description

In the crew rostering problem, a set of *activities* or *tasks*, such as pairings, training activities, ground duties and reserves, should be formed into rosters that are assigned to the crew members. Because of the large number of rosters that could be formed from the set of tasks, the crew rostering problem is solved using column generation.

This chapter describes how a general crew rostering problem can be formulated within the column generation framework. The rules concerning more than one roster are incorporated in the master problem, which is formulated over the set of all possible rosters. Hence, a column will correspond to a specific roster. The pricing problem will therefore consist of finding columns that correspond to rosters that fulfill the horizontal constraints related to the given problem.

3.1 The basic model and the master problem

Given a set $\mathcal{T}$ of tasks and a set $\mathcal{C}$ of crew members, the aim is to form a legal roster $\mathcal{R}_k \subset \mathcal{T}$ for each crew member $k \in \mathcal{C}$ such that all tasks are covered by exactly one crew member, i.e., such that

$$\mathcal{T} = \mathcal{R}_1 \cup \mathcal{R}_2 \cup \cdots \cup \mathcal{R}_{|\mathcal{C}|}, \tag{3.1}$$

and the objective is to minimize a linear function with respect to the cost c_k of each roster. Here, a roster is said to be legal if it fulfills the rules and regulations of the problem instance at hand.

For each crew member $k \in \mathcal{C}$, a large set of rosters can be formed and each roster can be modeled as a binary column vector $\boldsymbol{a}_j$, $j \in \mathcal{J}_k$, with dimension $m = |\mathcal{C}| + |\mathcal{T}|$. The first $|\mathcal{C}|$ rows determine to whom the roster described by the column can be assigned and the last $|\mathcal{T}|$ rows which tasks that belong to the roster. Hence, the column $\boldsymbol{a}_j$ can be decomposed into two parts

$$\boldsymbol{a}_j = \begin{bmatrix} \boldsymbol{e}_k \\ \boldsymbol{p}_j \end{bmatrix}, \qquad j \in \mathcal{J}_k, \tag{3.2}$$

where $\boldsymbol{e}_k \in \{0,1\}^{|\mathcal{C}|}$ is a unit column vector with 1 at position k, and $\boldsymbol{p}_j$ is a binary column vector. The set $\mathcal{J}_k$ contains indices corresponding to the columns belonging to crew member k, i.e.,

$$\mathcal{J} = \bigcup_{k \in \mathcal{C}} \mathcal{J}_k = \{1, 2, \dots, n\} \tag{3.3}$$

and

$$\mathcal{J}_k \cap \mathcal{J}_{k'} = \emptyset, \ \forall k \neq k', \tag{3.4}$$

where $n = |\mathcal{J}|$ is the total number of columns for all crew members. Let the $m \times n$-matrix A contain all column vectors $\boldsymbol{a}_j$, $j \in \mathcal{J}$, and let $\boldsymbol{x} \in \{0,1\}^n$ be the decision variables, where $x_j = 1$ if column $\boldsymbol{a}_j$ is chosen and 0 otherwise. Note that the number n of columns might be very large. In compact form, the crew rostering problem can be formulated as

$$\begin{aligned} \min \ & \boldsymbol{c}^\top \boldsymbol{x}, \\ \text{s.t.} \ & A\boldsymbol{x} = \mathbf{1}, \\ & \boldsymbol{x} \in \{0,1\}^n. \end{aligned} \tag{3.5}$$

The first $|\mathcal{C}|$ rows in the constraints matrix A make sure that each crew member is assigned to one roster and are hence called the *assignment constraints*. The *activity constraints* are defined by the last $|\mathcal{T}|$ rows of A and ensure that each task is covered exactly once.

By considering a simple example where each crew member $k \in \mathcal{C}$ have three rosters each, i.e. $n = 3|\mathcal{C}|$, the constraint matrix A can look like

$$A = \left[\begin{array}{cccc}
\overbrace{1 \ \ 1 \ \ 1}^{\text{Crew } 1} & \overbrace{0 \ \ 0 \ \ 0}^{\text{Crew } 2} & \overset{\cdots}{\cdots} & \overbrace{0 \ \ 0 \ \ 0}^{\text{Crew } |\mathcal{C}|} \\
0 \ \ 0 \ \ 0 & 1 \ \ 1 \ \ 1 & \dots & 0 \ \ 0 \ \ 0 \\
\vdots \ \ \vdots \ \ \vdots & \vdots \ \ \vdots \ \ \vdots & \ddots & \vdots \ \ \vdots \ \ \vdots \\
0 \ \ 0 \ \ 0 & 0 \ \ 0 \ \ 0 & \dots & 1 \ \ 1 \ \ 1 \\
1 \ \ 1 \ \ 0 & 1 \ \ 0 \ \ 1 & \dots & 0 \ \ 1 \ \ 0 \\
0 \ \ 1 \ \ 1 & 0 \ \ 0 \ \ 1 & \dots & 1 \ \ 0 \ \ 1 \\
\vdots \ \ \vdots \ \ \vdots & \vdots \ \ \vdots \ \ \vdots & \dots & \vdots \ \ \vdots \ \ \vdots \\
1 \ \ 0 \ \ 1 & 0 \ \ 0 \ \ 0 & \dots & 0 \ \ 1 \ \ 0
\end{array}\right]
\begin{array}{l}
\left.\vphantom{\begin{array}{c}1\\1\\1\\1\end{array}}\right\} |\mathcal{C}| \text{ assignment constraints} \\
\left.\vphantom{\begin{array}{c}1\\1\\1\\1\end{array}}\right\} |\mathcal{T}| \text{ activity constraints}
\end{array} . \tag{3.6}$$

The first $|\mathcal{C}|$ rows in each column describe which crew member the corresponding roster can be assigned to and the last $|\mathcal{T}|$ rows which tasks each roster includes. For example, the first column specifies a roster for the first crew member and includes the first task but not the second.

This basic model can be extended to include other vertical constraints, i.e. rules that concern more than one roster, by introducing additional constraints and generalizing the set partitioning problem in (3.5). One example is crew complement, where the model is extended to

$$\begin{aligned} \min \;& \boldsymbol{c}^\top \boldsymbol{x} \\ \text{s.t. } & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \in \{0,1\}^n, \end{aligned} \tag{3.7}$$

where $\boldsymbol{b}$ is a vector of positive integers. For the first $i = 1, 2, \ldots, |\mathcal{C}|$ elements, $b_i = 1$, since the constraint of only one assigned roster for each crew member $k \in \mathcal{C}$ still holds. The remaining $|\mathcal{T}|$ constraints specify the number of crew members that is needed on each task. For example, if task $t \in \mathcal{T}$ requires two crew members, then $b_{|\mathcal{C}|+t} = 2$. A description of further extensions, for example qualification constraints, constraints representing the experience of pilots, constraints defining that some crew members must fly together, and incompatibilities, are presented by Kohl and Karisch in [2].

To obtain an initial solution, it is common to include artificial variables in the problem (3.7), such that a solution can be formed even when one or several constraints are violated. The violations can for example be to leave a task unassigned in the solution or to leave one or several crew members unassigned. These types of violations have some real cost related to them, and for some problem instances, it might not be possible to assign all tasks or a better solution can be obtained when leaving certain tasks unassigned.

The problem in Eq. (3.7) forms the IMP and using the branch-and-price method presented in Chapter 2, a solution to the crew rostering problem can be obtained where the pricing problem consists of generating columns in A.

3.2 The pricing problem and the rule modeling language Rave

The legality and cost of a roster only depends on the crew member for which the roster can be assigned to. Therefore, the pricing problem in column generation for crew rostering can be solved for each crew member individually. For each crew member $k \in \mathcal{C}$, the problem of finding a column with negative reduced cost, i.e.

$$\min_{j \in \mathcal{J}_k} \{c(\boldsymbol{a}_j) - \bar{\boldsymbol{\pi}}^\top \boldsymbol{a}_j\} \tag{3.8}$$

should be solved. The dual variables $\bar{\boldsymbol{\pi}}$ are given by the solution to the RMP in the current iteration and all columns $\boldsymbol{a}_j$, $j \in \mathcal{J}_k$, fulfill the roster constraints which are, for instance, horizontal constraints regarding crew qualifications and rest times. Assuming that there exists legal rosters, the problem in (3.8), can be formulated over the set of all possible columns corresponding to both legal and illegal rosters, where the cost of an illegal roster is infinitely high.

At Jeppesen, the horizontal rules and regulations are expressed in the proprietary rule modeling language Rave. This enables the customers to easily maintain their own rules and a fast and correct modeling of the problem. Rave is used to check the attributes of a roster and to determine if it fulfills the constraints, i.e. is legal to assign to a crew member. It is also used to obtain the cost coefficients in the pricing problem in (3.8).

The search for columns should be performed using information that can be provided by Rave. The evaluation of a roster will only result in a yes/no answer – if the roster is legal or not – and the value of the cost of assigning the roster to the given crew member would be.

The objective of the pricing problem (3.8) consists of a linear part with respect to the elements in the column vector $\boldsymbol{a}_j$, and therefore with respect to the tasks. However, the cost function $c(\cdot)$ in (3.8) is implemented in Rave, and can be considered as a nonlinear, unknown function which complicates solving the pricing problem.

Because of the large number of possible columns, both legal and illegal, it is not practical to randomly generate columns and check legality and cost, since the probability of even finding a legal column might be low. Although all rules and constraints regarding the rosters are considered unknown, the problem has a partly known structure which means that a completely random search is not necessary.

Each task is scheduled to occur at some given time and therefore, the problem of forming a legal column can be seen as finding a path in a network where each task is represented by a node. By solving the pricing problem using such a graph, a roster with two tasks overlapping in time is never formed.

However, there is no guarantee that all columns corresponding to paths in the network will be legal. On the contrary, it is very likely that a large number of columns will still be illegal. Some illegality can be handled by adding resource constraints, but these would require prior knowledge of the rostering problem instance at hand; this knowledge might not be available, and still does not guarantee that a legal roster is found. Also, solving a shortest path problem with resource constraints is NP-complete [14], and since the number of tasks can be very high, solving such a problem would require some heuristic approach.

Another difficulty is how to set the arc costs in the network and thereby find useful paths. As mentioned, the cost function can be considered to be a nonlinear function which means that linear arc costs cannot be formed. Even if the constraints would be available, and if arc costs could be derived, the resulting problem is a nonadditive shortest path problem, which is time consuming to solve, and therefore some heuristic requiring an explicit form for the cost function would have to be applied.

Finally, the problem might be very large-scale; there can be thousands of tasks that could be scheduled for a crew member. This also adds to the computational demand of solving the problem.

The currently used solution methods for the pricing problem at Jeppesen is to solve a constrained shortest path problem and to use a local search heuristic. Alternative approaches are to focus more on the black-box part of the problem.

4 Binary particle swarm optimization for the pricing problem

The first solution approach was to use binary particle swarm optimization [27] to generate new columns in an iterative process guided by both randomness and previously known good columns. Using this method, no assumptions regarding the structure of the problem had to be made.

Stochastic methods such as particle swarm optimization and the genetic algorithm [28, p. 209] has been popular both in academia and the industry due their ability to solve highly nonlinear, mixed integer optimization problems that can be found in complex engineering systems and because of their ease of implementation. Compared to the genetic algorithm, the particle swarm optimization has been claimed to have the same effectiveness but with significantly lower computational cost [29].

In each iteration of the binary particle swarm optimization, a set of columns were modified using probabilities, where the probability of assigning a task was higher if the task had been included in a column with low reduced cost in a previous iteration. This meant that tasks belonging to a previously found roster with low reduced cost were likely to be included in the new roster but because of randomness completely new columns might be found by the model. In this way, no assumptions regarding the cost structure was made except that tasks belonging to a roster with low reduced cost are beneficial to assign.

In this chapter the binary particle swarm optimization algorithm is presented together with a description of how the method can be used to generate columns for the pricing problem in column generation for crew rostering.

4.1 Binary particle swarm optimization

Particle swarm optimization (PSO) is a population-based optimization technique for solving continuous optimization problems. PSO was introduced by Kennedy and Eberhart in 1995 [27] and was inspired by the swarming behavior found in bird flocks and fish schools. The given problem is solved by iteratively improving a set of candidate solutions, here noted as particles, by moving them towards previously known good solutions. Each particle is associated with a position and a velocity in the search space. In each iteration, the particles are guided towards a linear combination of their previously known local best position and the best position found by the entire swarm.

In *binary particle swarm optimization (BPSO)*, proposed by Kennedy and Eberhart [30] to solve binary optimization problems, the velocity is passed through a continuous transfer function taking values in the interval [0,1]. The function value is then used as a probability of what position the particle will have in the next iteration.

4.1.1 The general algorithm

Let N be the number of particles in the swarm and let $\mathbf{p}^i = [p_1^i, p_2^i, \ldots, p_d^i]$ be the position of particle $i = 1, 2, \ldots, N$, in a d-dimensional space and let $\mathbf{v}^i = [v_1^i, v_2^i, \ldots, v_d^i]$ be the particle's corresponding velocity.

First, the position and velocity of each particle are initialized. The objective function $f(\cdot)$ to be minimized is evaluated using the position of each particle. The best position found by each particle i, $\boldsymbol{p}^i_{\text{pBest}}$, as well as the best position found by the entire swarm, $\boldsymbol{p}_{\text{gBest}}$, are updated using the function values calculated using the particles' current positions, i.e.

$$\text{if } f(\boldsymbol{p}^i) < f(\boldsymbol{p}^i_{\text{pBest}}), \text{then } \boldsymbol{p}^i_{\text{pBest}} \leftarrow \boldsymbol{p}^i, \tag{4.1}$$

$$\text{if } f(\boldsymbol{p}^i) < f(\boldsymbol{p}_{\text{gBest}}), \text{then } \boldsymbol{p}_{\text{gBest}} \leftarrow \boldsymbol{p}^i. \tag{4.2}$$

Using the best positions found so far, the velocities for each particle $i = 1, 2, \ldots, N$ are updated according to

$$v_t^i \leftarrow w v_t^i + c_1 q(p^i_{\text{pBest},t} - p_t^i) + c_2 r(p_{\text{gBest},t} - p_t^i), \;\; t = 1, 2, \ldots, d, \tag{4.3}$$

where c_1 and c_2 are constants known as the *cognitive component* and the *social component*, q and r are random numbers in the interval [0,1] and w is the so-called *inertia weight* used to handle the trade-off between exploration and exploitation. If $w > 1$, exploration is favored and if $w < 1$ exploitation will be prioritized by drawing the particles towards the best known positions. A common approach is to favor exploration at the beginning of the algorithm such that a more global search is performed and then towards the end focus the search to previously known good areas. This is done by reducing the inertia weight w with a constant factor in each iteration until a lower bound has been reached. Common choices of parameters are presented by Wahde [31].

The velocities should be limited to the interval $[-v_{\max}, v_{\max}]$. If the updated velocity exceeds this interval, it is set to $-v_{\max}$ or $v_{\max}$ depending on its sign.

Using an appropriate transfer function, the particle will be more prone to move towards a previously known good solution but with some probability move in another direction. The positions of the particles are updated probabilistically using the velocities passed through the transfer function, which will be explained in the next section, Sec. 4.1.2. The velocity is then updated using the new position according to the update rule in (4.3). New probabilities are then obtained by evaluating the transfer function using the new velocity. This process of calculating probabilities, updating positions and velocities is repeated until a stopping criterion is met. The stopping criterion could, for example, be when a maximum predefined number of iterations have been performed.

4.1.2 Different transfer functions

The transfer function transforms the velocity into a probability used to determine the position in the next iteration. In the original BPSO, Kennedy and Eberhart [30] use the transfer function

$$\sigma(v_t^i) = \frac{1}{1+e^{-v_t^i}}, \tag{4.4}$$

where the position in the next iteration is updated according to

$$p_t^i \leftarrow \begin{cases} 0 \text{ if } r < \sigma(v_t^i), \\ 1 \text{ otherwise.} \end{cases} \tag{4.5}$$

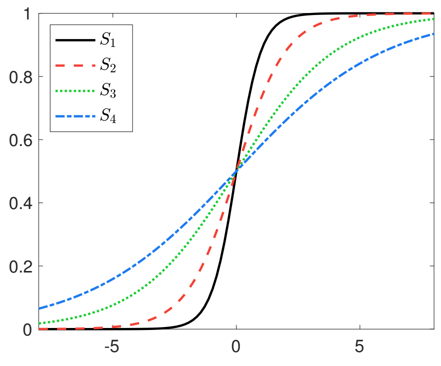
where r is a random number in the interval $[0, 1]$.

Since the original method was proposed, several different choices of transfer functions have been proposed. In the article by Mirjalili and Lewis [32], a study of different functions is presented. The authors describe several concepts that need to be considered when selecting the transfer function. Since the function is used as a probability, it should be bounded by the interval [0,1]. A high value of the absolute value of the velocity indicates that the particle is far from the best solution found, and therefore it should be likely that the particle switch position in the next iteration. The opposite holds for a low value of the absolute value of the velocity. Also, if a particle is moving away from the best solution, the probability of changing the position should increase. Conversely, if a particle moves towards the best solution, the probability of change should decrease.

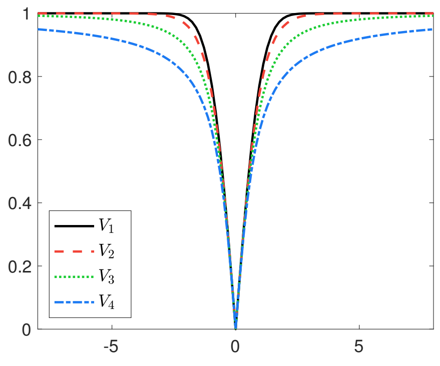
Mirjalili and Lewis introduce two families of transfer functions, namely the *S-shaped* and *V-shaped transfer functions*, where the S-shaped functions take low values for velocities close to $-v_{\max}$ and high values for velocities close to $v_{\max}$ while the V-shaped functions take high values when the absolute value of the velocity is close to $v_{\max}$. The function in (4.4) is an example of an S-shaped function. Other examples of S- and V-shaped functions are listed in Tab. 4.1 and graphically illustrated in Fig. 4.1.

Table 4.1: Examples of S- and V-shaped transfer functions. The function erf$(\cdot)$ is the error function, $\text{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^{x} e^{-t^2} \text{d}t$.

S-shaped	**V-shaped**
$S_1(v) = \frac{1}{1+e^{-2v}}$	$V_1(v) = \lvert\text{erf}(\frac{\sqrt{\pi}}{2}v)\rvert$
$S_2(v) = \frac{1}{1+e^{-v}}$	$V_2(v) = \lvert\tanh(v)\rvert$
$S_3(v) = \frac{1}{1+e^{-\frac{v}{2}}}$	$V_3(v) = \left\lvert\frac{v}{\sqrt{1+v^2}}\right\rvert$
$S_4(v) = \frac{1}{1+e^{-\frac{v}{3}}}$	$V_4(v) = \left\lvert\frac{2}{\pi}\arctan\left(\frac{\pi}{2}v\right)\right\rvert$



(a) S-shaped transfer functions.



(b) V-shaped transfer functions.

Figure 4.1: The different transfer functions listed in Tab. 4.1.

For the S-shaped transfer functions, the update rule is according to the one presented in Eq. (4.5). For the V-shaped functions, the particle's position is instead updated according to

$$p_t^i \leftarrow \begin{cases} \text{complement}(p_t^i) \text{ if } r < \sigma(v_t^i), \\ p_t^i \text{ otherwise.} \end{cases} \tag{4.6}$$

where complement$(p) = 1 - p$ and r is a random number in the interval $[0, 1]$.

4.2 An algorithm for finding good solutions to the pricing problem using binary particle swarm optimization

Recall from Sec. 3.2 that the pricing problem consists of finding columns with negative reduced cost, where a column corresponds to a roster which is formed from a given set of tasks. Since a column is a binary vector, BPSO can be used to search for a solution.

The pricing problem is solved separately for each crew member. Given crew member $k \in \mathcal{C}$, the set of tasks $\mathcal{T}_k \subset \mathcal{T}$ that could be assigned to this crew member can be obtained.

During each BPSO iteration, each particle is associated with a column $\boldsymbol{a}_j, j \in \mathcal{J}_k$, where $\mathcal{J}_k$ is the set of all column indices for crew member k. The position of the particle is determined by the tasks that are included in the column. Therefore, if the position changes in the next iteration, the particle is associated with a new column. The position vector of a particle is described by the binary vector $\boldsymbol{p}_j$ which corresponds to the last $|\mathcal{T}|$ elements of $\boldsymbol{a}_j$; see Chapter 3, (3.2). Note that, the first $|\mathcal{C}|$ elements describe which crew member the corresponding roster can be assigned to, and are therefore identical for all particles in all iterations. Thus, this information does not need to be included in the position of the particles. Denote the position of particle $i = 1, 2, \ldots, N$ by $\boldsymbol{p}^i \in \{0, 1\}^{|\mathcal{T}|}$ and its velocity by $\boldsymbol{v}^i \in \mathbb{R}^{|\mathcal{T}|}$.

The first step is to initialize the positions of the N particles. The column $\boldsymbol{a}_j$ with the highest primal value x_j is used as one of the starting positions, and therefore, the search for new columns is influenced by the best known column and the algorithm can be seen as a form of local search [33]. The remaining $N - 1$ particles/columns are generated randomly from the set $\mathcal{T}_k$ of tasks. Each element of the velocity vector for each particle is initialized uniformly in $[-v_{\max}, v_{\max}]$. Then, until the maximum number of iterations is reached, the algorithm searches for new columns.

In each iteration, each column is evaluated in Rave and the reduced cost corresponding to each particle is computed. If the column has a negative reduced cost, it is saved to the set of columns that might enter the RMP. The best column found by each particle i, $\boldsymbol{p}^i_{\text{pBest}}$, as well as the best column found by all particles, $\boldsymbol{p}_{\text{gBest}}$, are updated by comparing the reduced costs of the previously known best positions with the reduced cost corresponding to each particle's current position. The velocities are then updated according to (4.3) and each element in each velocity vector is restricted to $[-v_{\max}, v_{\max}]$, i.e.

$$\begin{aligned} &\text{if } v^i_t > v_{\max}, && \text{then } v^i_t \leftarrow v_{\max} \\ &\text{if } v^i_t < -v_{\max}, && \text{then } v^i_t \leftarrow -v_{\max}. \end{aligned} \tag{4.7}$$

In the final step, the particles are given new positions. A V-shaped transfer function is chosen, so that each particle is encouraged to stay in its current position unless the absolute value of the velocity is high, i.e., close to 1. Using the update rule in (4.6) the particles are given new positions.

With each new assignment, there is a risk of creating an illegal roster. Also, because of the dimension of practical problem instances, it is highly likely that some tasks with a low assignment probability will get assigned. Since the number of tasks that can be assigned to a roster is limited, too many assignments cannot be made without creating an illegal roster. Therefore, to prevent illegal rosters from being formed, a legality check is performed for each new task $t \in \mathcal{T}_k$ not belonging to the previous roster, i.e. where the position p^i_t should change from 0 to 1.

Algorithm 1 `updatePositionUsingLegalityCheck`($\boldsymbol{p}, \boldsymbol{v}$,`cost`)

```
 1: $\boldsymbol{\sigma} \leftarrow$ computeProbabilityOfChange($\boldsymbol{v}$)
 2: $\mathcal{T}_{\text{new}} \leftarrow [\,]$          // Set of tasks to try to assign
 3:
 4: for all tasks $t \in \mathcal{T}_k$ do
 5:     if rand $< \sigma_t$ then
 6:         $\bar{p_t} \leftarrow$ complement($p_t$)
 7:         if $\bar{p_t} = 1$ then
 8:             $\mathcal{T}_{\text{new}} \leftarrow$ addTask($\mathcal{T}_{\text{new}}$,$t$)
 9:         else
10:             $p_t \leftarrow$ deassign($t$)
11:         end if
12:     else
13:         $p_t \leftarrow p_t$             // Do nothing
14:     end if
15: end for
16:
17: couldAssign $\leftarrow$ FALSE
18: for all tasks $t \in \mathcal{T}_{\text{new}}$ do
19:     if tryAssign($t$) == Successful then
20:         $p_t \leftarrow 1$
21:         couldAssign $\leftarrow$ TRUE
22:     else
23:         $p_t \leftarrow 0$
24:     end if
25: end for
26:
27: if couldAssign then
28:     cost$\leftarrow$ calculateReducedCost($\boldsymbol{p}$)
29: else
30:     if !isLegal($\boldsymbol{p}$) then
31:         cost $\leftarrow$ highCost
32:     else
33:         cost$\leftarrow$ calculateReducedCost($\boldsymbol{p}$)
34:     end if
35: end if
36: return $\boldsymbol{p}$, cost
```

The algorithm for the position update using legality check is presented in Algorithm 1. The new position of a particle was computed using the transfer function as a probability of change. Since the probability of a roster remaining legal after after a task has been removed/deassigned from the roster is high, a change in position from 1 to 0 is performed without checking for legality in Rave. Note that the corresponding roster after the deassignment-step is certain to be possible to extend to a legal roster by adding one or several tasks. However, since legality cannot be assumed for the roster formed when assigning a new task, for each new assignment, i.e., every time a particle's position would change from 0 to 1, the legality of the new roster is checked by Rave using the function `isLegal`.

For each task $t \in \mathcal{T}_k$, if the position p_t^i of particle i, changes from 0 to 1, the task is added to the set of new tasks to try to assign. Since deassigning a task might make an assignment possible, the deassignment of tasks occurs before the assignment-step. All tasks for which legality should be checked are added to the set $\mathcal{T}_{\text{new}} \subseteq \mathcal{T}_k$. Then, for each task $t \in \mathcal{T}_{\text{new}}$, the algorithm tries to assign it by calling Rave. The tasks are checked in the order that they were added to the set $\mathcal{T}_k$. If successful, p_t^i is set to 1, else remains 0.

After the assignment-step, the reduced cost is calculated using the position of the particle using the function `calculateReducedCost` which evaluates the cost function in Rave together with (3.8). Note that, if no new tasks could be added to the roster, the updated position of the particle might correspond to an illegal roster. Therefore, before calculating the reduced cost, the legality of the updated position is checked using the function `isLegal` if the boolean `couldAssign` is false, i.e. no new assignments have been made. In the unlikely scenario that an illegal roster has been formed, a high reduced cost is set to make sure that the best position found by the particle as well as the best position found by the entire swarm will not correspond to an illegal roster.

The complete BPSO for generating new columns is presented in Algorithm 2. The update-functions `updateParticleBest` and `updateSwarmBest` use the rules in (4.1) and (4.2) and the `updateVelocity` uses the rule in (4.3).

Algorithm 2 `solvePricingUsingBPSO()`

```
 1: alternativeColumns ← [ ]          //Columns with negative reduced cost
 2: for all particles $i$ do
 3:     [$\boldsymbol{p}^i, \boldsymbol{v}^i$, cost$^i$] ← initialize()
 4: end for
 5:
 6: repeat
 7:     for all particles $i$ do
 8:         if cost$^i$ $< 0$ then
 9:             alternativeColumns ← saveColumnToRMP($\boldsymbol{p}^i$)
10:         end if
11:         $\boldsymbol{p}^i_{\text{pBest}}$ ←updateParticleBest(cost$^i$, $\boldsymbol{p}^i_{\text{pBest}}$)
12:         $\boldsymbol{p}_{\text{gBest}}$ ←updateSwarmBest(cost$^i$, $\boldsymbol{p}_{\text{gBest}}$)
13:     end for
14:     for all particles $i$ do
15:         $\boldsymbol{v}^i$ ← updateVelocity(pBest$^i$, gBest, $\boldsymbol{p}^i$)
16:         $\boldsymbol{p}^i$, cost$^i$ ← updatePositionUsingLegalityCheck($\boldsymbol{p}^i, \boldsymbol{v}^i$, cost$^i$)
17:     end for
18: until maxIter
19: return alternativeColumns
```

5 Surrogate modeling to solve the pricing problem

The second approach was to create a nonlinear surrogate function using *radial basis functions* that could be used instead of the black-box cost function from Rave to search for columns with negative reduced costs. Using the set of initial columns a surrogate function was created using interpolation.

The resulting surrogate function was a nonlinear function of binary variables. To find columns with negative reduced cost, two different solution methods were implemented. In the first method, LinSurMod, the surrogate function was linearized such that edge costs in a network could be obtained using the fitted parameters to the surrogate function. By solving the corresponding shortest path problem, a candidate solution was found as the shortest path in the network. By adding the found column to the set of sampled columns and re-fitting the model parameters, slightly different edge costs were obtained, defining a new shortest path problems. By continuing this iterative procedure, new columns were found in each iteration.

The second method, SurMod, was to approximate the objective function of the pricing problem using the nonlinear surrogate function. For this approach, one of the purposes was to investigate if the nonlinear surrogate function could capture the characteristics of the true cost function and what effect the linearization has. Since a BPSO had already been implemented, a slightly modified version of Algorithm 2 was used to minimize the surrogate function heuristically. Here, the position of each particle was evaluated with respect to the surrogate function. In an iterative process similar to the method using the linearized surrogate function, new columns were generated in each iteration.

5.1 Surrogate modeling with radial basis functions

A *surrogate model*, also known as a *response surface model*, approximates the behavior of a complex system. The surrogate function is constructed based on a limited set of data points from the black-box function, and therefore mimics the behavior of the underlying model [34]. Surrogate modeling is often used if function evaluations are computationally expensive. Instead of having to evaluate the expensive black-box function, the surrogate can be used to search for an optimal solution. However, the approach can also be useful for less expensive functions since it offers an explicit expression that can be useful when constructing a solution method.

There are different kinds of surrogate models based on for example polynomials, kriging, or radial basis functions (RBF), where the two latter are more suitable for highly nonlinear and multidimensional functions [35]. In this thesis, RBFs were chosen to construct the surrogate model.

An RBF is a function ϕ,for which the value at each point depends only on the distance between that point $\boldsymbol{x}$ and a target point $\boldsymbol{x}_0$, i.e., $\phi = \phi(r)$ where $r = ||\boldsymbol{x} - \boldsymbol{x}_0||$ and $||\cdot||$ is usually the Euclidean distance. There are different types of radial basis functions [35], but the most common choices are

- linear: $\phi(r) = r$,
- cubic: $\phi(r) = r^3$,
- thin plate spline: $\phi(r) = r^2 \log(r)$,
- multiquadratic: $\phi(r) = \sqrt{r^2 + \gamma^2}$, where $\gamma > 0$ is a constant and
- Gaussian: $\phi(r) = e^{-\gamma r^2}$, where $\gamma > 0$ is a constant.

Given a set of initial points, $S = \{\boldsymbol{p}^1, \boldsymbol{p}^2, \dots, \boldsymbol{p}^n\}$, where $\boldsymbol{p} \in \mathbb{R}^d$ and $\boldsymbol{p}_i \neq \boldsymbol{p}_j$ for all $i \neq j$, an RBF interpolant can be written as

$$s(\boldsymbol{p}) = \sum_{i=1}^{n} \lambda_i \phi(||\boldsymbol{p} - \boldsymbol{p}_i||). \tag{5.1}$$

As described by Rocha [36], a unique interpolant is guaranteed for the multiquadratic and the Gaussian RBF since it can be shown for these cases that the matrix Φ, with elements $\phi_{ij} = \phi(||\boldsymbol{p}_i - \boldsymbol{p}_j||)$, is nonsingular even if the input points $\boldsymbol{p}_i, i = 1, 2, \dots, n$ are poorly distributed. However, for the cubic and thin plate spline RBFs, Φ might be singular. The remedy is to add low-degree polynomials to the interpolant in (5.1) which gives the following

$$s(\mathbf{p}) = \sum_{i=1}^{n} \lambda_i \phi(||\boldsymbol{p} - \mathbf{p}_i||) + \boldsymbol{\beta}^\top \boldsymbol{p} + \alpha. \tag{5.2}$$

In matrix formulation, the interpolation equations can be formulated as

$$\begin{bmatrix} \Phi & P \\ P^\top & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda} \\ \tilde{\boldsymbol{\beta}} \end{bmatrix} = \begin{bmatrix} F \\ \mathbf{0} \end{bmatrix} \tag{5.3}$$

where

$$P = \begin{bmatrix} \boldsymbol{p}_1^\top & 1 \\ \boldsymbol{p}_2^\top & 1 \\ \vdots & \vdots \\ \boldsymbol{p}_n^\top & 1 \end{bmatrix}, \ \boldsymbol{\lambda} = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{bmatrix}, \ \tilde{\boldsymbol{\beta}} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_d \\ \alpha \end{bmatrix}, \ F = \begin{bmatrix} f(\boldsymbol{u}_1) \\ f(\boldsymbol{u}_2) \\ \vdots \\ f(\boldsymbol{u}_n) \end{bmatrix}. \tag{5.4}$$

As long as $\operatorname{rank}(P) = d + 1$ the linear system will have a unique solution and a unique RBF interpolant to the function at the points in S can be obtained [36]. The parameters $\boldsymbol{\lambda}$ and $\tilde{\boldsymbol{\beta}}$ to the surrogate model can for example be computed using LU factorization or LDL$^\top$ factorization [37].

The complete algorithm consists of first generating a set of initial points. Then, in an iterative process, construct a surrogate model, which is used to search for new candidate points that should be incorporated in the set S to enhance the fit of the model and thereby get closer to finding the optimal solution [35]. An example of solving mixed integer nonlinear optimization problems is presented by Holmström et al. [38] and an example of a nonlinear integer optimization problem solved using surrogate modeling with radial basis functions is given by Müller et al. [39].

The final step is finding a candidate point, i.e. the next point for evaluation, which is used to refine the fit of the surrogate function. The point is chosen as the one that maximizes (or minimizes) a specific merit function. Vu et al. [35] lists several different approaches to find candidate points, for example a class of merit functions that use the standard error in statistical models. Another type of merit function is to use the surrogate function itself to search for new points. This works well if the model is reliable, but on the other hand can be misleading if the model is biased. However, since the surrogate function itself often is relatively simple to optimize, the resulting subproblem will be easy to solve.

5.2 The algorithm for solving the pricing problem using model fitting

Given a crew member $k \in \mathcal{C}$ and a set of tasks $\mathcal{T}_k \subseteq \mathcal{T}$ that could be assigned to k, a column $\boldsymbol{a}_j, j \in \mathcal{J}_k$, with negative reduced cost should be found, where $\mathcal{J}_k$ is the set of all column indices for crew member k. Same as for the BPSO algorithm described in the previous chapter, since the crew member is given, the first $|\mathcal{C}|$ elements in $\boldsymbol{a}_j$ are fixed to the unit vector $\boldsymbol{e}_k$ and only the last $|\mathcal{T}|$ elements may vary.

From a set of initial columns, a surrogate function was obtained using interpolation. Since the initial columns were chosen from the columns included in the RMP, our hope was that the model would be able to capture some important aspects of legal rosters. Therefore, by simply using the surrogate function to find promising columns the search would be located more closely towards the initial set of columns. By adding a new column and fitting the model to the new set of columns, a slightly different model was formed. By continuing this iterative process of fitting a model, finding a candidate solution using the surrogate function, adding the corresponding column to the set of sampled columns and refitting the model, a search for new columns was performed.

The problem of finding a candidate solution was solved in two different ways. The first approach was to solve a shortest path problem where the edge costs in the network were given by a linear approximation of the surrogate function. The second approach was to instead minimize the surrogate function using a BPSO algorithm.

5.2.1 Task selection and creating an initial set of columns

Finding the initial columns is an important step of the algorithm. These columns are used to construct the initial surrogate model and therefore provide the first information about the black-box function. For example, if all points were located close to each other or close to the same local optimum, the model would be biased to that region and might not be able to find a global optimum.

The initial set of columns is formed partly from the set of columns in the RMP corresponding to the crew member for which the pricing problem is to be solved. By simply using the columns from the RMP, the initial set is guaranteed to be legal and less computational time has to be spent on generating new legal columns. Also, since these columns have been found by solving pricing problems in previous iterations, it is reasonable to assume that these columns have some beneficial characteristics which might not be obtained by generating all initial columns randomly.

The number of tasks in $\mathcal{T}_k$ could be very large, and since the matrix P must have a rank equal to the number of tasks plus one, the linear system of equation (5.3) can become very large. Therefore, the set of tasks $\mathcal{T}_k$ is reduced for the surrogate modeling pricing methods to prevent the linear system from becoming too large and to reduce the number of initial columns required to solve the system (5.3) of equations. This is a significant difference compared to the BPSO-method, which uses the entire search space, i.e. all tasks in $\mathcal{T}_k$, to find columns. By selecting a subset of tasks $\mathcal{T}'_k \subseteq \mathcal{T}_k$, only columns with tasks belonging to $\mathcal{T}'_k$ are allowed to be assigned and hence fewer initial samples are required to ensure the rank of the matrix P. On the other hand, by omitting tasks from being assigned, the method will be unable to find certain columns and the performance of the surrogate modeling approach will therefore depend on the selection of tasks.

A threshold for the number of tasks is chosen and using the columns from the RMP enables the first selection. Only tasks that are included in at least one column from the RMP are selected. This is done both to reduce the dimension of the problem but also to make sure that the search is located more closely to already sampled columns which are legal by definition.

If this selection still results in a set of tasks larger than the chosen threshold, more tasks has to be removed and hence also some of the columns obtained from the RMP. To perform this selection, the heuristic in Algorithm 3 is used. The tasks to include are selected with respect to the best columns in the RMP. First, the best column is found and all tasks that are included in the column are included in $\mathcal{T}'_k$. Then, the tasks from the second best column are included in $\mathcal{T}'_k$, then from the third best, and so on. Tasks are added to $\mathcal{T}'_k$ until the threshold is passed. Since there can be other columns from the RMP that only contain tasks in $\mathcal{T}'_k$, these are found

Algorithm 3 `selectTasksAndGetInitialColumns()`

```
1: $S_{\text{RMP}} \leftarrow$ getColumnsFromRMP()
2: $\mathcal{T}_{\text{RMP}} \leftarrow$ removeAllTasksWithNoColumnFromRMP($\mathcal{T}_k, S$)
3:
4: if $|\mathcal{T}_{\text{RMP}}| >$threshold then
5:     $\boldsymbol{a} \leftarrow$ getBestColumnFromRMP($S_{\text{RMP}}$)
6:     repeat                                        // Task selection
7:         $\mathcal{T}_k' \leftarrow$ insertTasksFromColumns($\boldsymbol{a}$)
8:         $S \leftarrow$ addColumnToInitialSet($\boldsymbol{a}$)
9:         $\boldsymbol{a} \leftarrow$ getNextBestColumn($S_{\text{RMP}}, \boldsymbol{a}$)
10:    until $|\mathcal{T}_k'| >=$threshold
11:    $S \leftarrow$ addAllColumnsFromRMPThatContainChosenTasks($\mathcal{T}_k', S_{\text{RMP}}$)
12: else
13:    $\mathcal{T}_k' \leftarrow \mathcal{T}_{\text{RMP}}$
14:    $S \leftarrow S_{\text{RMP}}$
15: end if
16:
17: if $|S| < |\mathcal{T}_k'|$ then                        // Add random columns
18:    $S \leftarrow$ addRandomColumns($\mathcal{T}_k'$)
19: end if
20:
21: for all $t \in \mathcal{T}_k'$ do                        // Ensure rank of $P$
22:    $\boldsymbol{p} \leftarrow$ createColumn($t$)
23:    $S \leftarrow$ insertColumn($\boldsymbol{p}$)
24: end for
25: $S \leftarrow$ insertEmptyColumn()
26:   return $S, \mathcal{T}_k'$
```

and included in the set S (`addAllColumnsFromRMPThatContainChosenTasks`).

To make sure that the surrogate model is based on many columns, randomly generated columns are also included in S. If the number of columns in S originating from the columns in the RMP, was less than the number of tasks times two, randomly generated columns were added to S. Finally, to ensure the rank of the matrix P, columns corresponding to rosters with exactly one task assigned are added to the set of initial columns together with a column corresponding to an empty roster. In this way, the last $|\mathcal{T}_k'|+1$ rows in P form an identity matrix and $\text{rank}(P) = |\mathcal{T}_k'|+1$ is guaranteed. Note that all sampled points should be unique, and hence if a column has already been sampled, it is not duplicated in S.

5.2.2 Computing the parameters of the surrogate function

To find the model parameters, the linear system for the RBF interpolant in (5.3) has to be solved. Since the initial columns in S are chosen such that $\text{rank}(P) = |\mathcal{T}_k'|+1$, the linear system of equations is guaranteed to have a unique solution. To find the parameters, the C++ library Eigen [40] was used to perform an *LU* factorization with partial pivoting using the class `PartialPivLU`.

5.2.3 Finding candidate solutions using a linearization of the surrogate function

Since the problem of forming columns from a set of tasks can be easily modeled in a graph, and thereby exclude some illegal columns, which for example include tasks that overlap in time, a shortest path approach was implemented using a linear approximation of the surrogate function. The surrogate model consists of a nonlinear part, and therefore cannot be directly modeled as a function of edge costs in a network. By instead considering an approximation of the surrogate function that is linear with respect to the Hamming distance of tasks differing between the points $\boldsymbol{p}$ and $\boldsymbol{p}_l, l = 1, 2, \dots, |S|$, i.e.,

$$\bar{s}(\boldsymbol{p}) = \sum_{l=1}^{n} \lambda_l ||\boldsymbol{p} - \boldsymbol{p}_l||_2^2 + \boldsymbol{\beta}^\top \boldsymbol{p} + \alpha = \hat{\boldsymbol{b}}^\top \boldsymbol{p} + \hat{a}, \tag{5.5}$$

linear edge costs can be extracted; see Appendix A. The number of tasks that can be assigned to a roster is typically quite small and therefore the number of differing tasks in two columns can be assumed to be small as well.

Also, the surrogate function should only be able to rank columns in the correct order with respect to reduced cost, and therefore, is not required to return the true reduced cost. By performing this linearization, and assuming that the mutual order of columns is still preserved, then the best column according to linear approximation also has a low value according to the nonlinear surrogate function.

The graph over all tasks in $\mathcal{T}_k'$ is constructed in such a way that all pairwise connections are legal, using the legality check in Rave. Edge costs are formed from the approximation of the surrogate function. The candidate solution, i.e. the new column, is then given by the shortest path in the constructed network. Since only pairwise connections have been checked for legality, the column found may correspond to an illegal roster and therefore the legality of each new column has to be checked in Rave before the true reduced cost can be calculated. If a column is illegal, the reduced cost is set to a high value.

To solve the shortest path problem, the Bellman-Ford algorithm is used [41, pp. 87–126.] since the edge costs in the constructed network can be positive or negative. The algorithm computes the shortest path from a given node to all other nodes. Note that as a part of the Bellman-Ford algorithm, a search for negative cycles in the graph is performed. Since the network with tasks as nodes is time-based, and therefore has no cycles, this step is, however, not necessary to perform.

Finding a new column using a linearization of the surrogate function is presented in Algorithm 4. First, the parameters $\hat{\boldsymbol{b}}$ and $\hat{a}$ of the linear approximation in (5.5) are obtained, then edge costs are set up using these parameters, and the corresponding shortest path problem is solve.

Algorithm 4 `findBestColumnAccordingToLinearSurrogate`($\mathcal{G}, \lambda, \boldsymbol{b}, a$)

```
1: $[\hat{\boldsymbol{b}}, \hat{a}] \leftarrow$ getLinearApproximation($\lambda, \boldsymbol{b}, a$)
2: $\mathcal{G} \leftarrow$ setupEdgeCosts($\hat{\boldsymbol{b}}, \hat{a}$)
3: $\boldsymbol{p} \leftarrow$ solveShortestPath($\mathcal{G}$)
4: return $\boldsymbol{p}$
```

5.2.4 Finding candidate solutions by minimizing the nonlinear surrogate function

An alternative to the linearization of the surrogate function for finding a new column, is to minimize the nonlinear surrogate function. Solving a nonlinear optimization problem with binary requirements on the variables is nontrivial and since a BPSO algorithm had already been implemented (see Chapter 4) we chose to reuse this implementation. Also, the approach to minimize a nonlinear surrogate function was chosen mainly to investigate its potential of capturing the true cost function.

The BPSO algorithm described in Chapter 4 was slightly modified. The position of each particle was evaluated using the surrogate function instead of (3.8) where $c(\cdot)$ is the true cost function in Rave, and an S-shaped transfer function was used. Instead of checking legality by calling Rave, and not assigning a task if the corresponding roster would turn out to be illegal, the constructed network, $\mathcal{G}$, used in the shortest path problem, was used to ensure that the assigned tasks would correspond to a path in the network in which each pairwise connection is legal; see Algorithm 5.

All tasks that the algorithm wants to assign to the new position are stored in a list. By directly adding these tasks to the column, the resulting column might correspond to a path which is not possible to form in the graph. In other words, this means that at least two tasks in the roster is not pairwise legal. Therefore, the tasks to assign are not added directly to the column. Instead, the first task in the list is assigned. Then, the next task in the list is assigned if there is a connection between this task and the one previously assigned, otherwise the second next task is checked and so on, until all tasks have been investigated.

The BPSO used to minimize the surrogate function is found in Algorithm 6. An important difference from the BPSO method presented in Chapter 4, is that illegal columns might be generated, since paths that correspond to illegal rosters may be formed in the network. The update-functions in the algorithm `updateParticleBest`, `updateSwarmBest` and `updateVelocity` are identical to the ones in Algorithm 2.

Algorithm 5 `updatePositionUsingGraph`$(\mathcal{G}, p, \boldsymbol{v})$

```
 1: $\boldsymbol{\sigma} \leftarrow$ computeProbabilityOfChange($\boldsymbol{v}$)
 2: tasksToAssign $\leftarrow$ [ ]         // Set of tasks to try to assign
 3:
 4: for all tasks $t \in \mathcal{T}_k'$ do
 5:     if rand $\geq \sigma_t$ then
 6:         tasksToAssign $\leftarrow$ tasksToAssign.add($t$)
 7:     end if
 8: end for
 9:
10: $\boldsymbol{p} \leftarrow$ empty
11: $\boldsymbol{p} \leftarrow$ assignTask($\boldsymbol{p}$, tasksToAssign[0])
12: latestAddedTask $\leftarrow$ tasksToAssign[0]
13: for all tasks $t \in$ tasksToAssign except first task do
14:     if tasksAreConnected($\mathcal{G}$, latestAddedTask, $t$) then
15:         latestAddedTask $\leftarrow t$
16:         $\boldsymbol{p} \leftarrow$ assignTask($\boldsymbol{p}$, $t$)
17:     end if
18: end for
19: return $\boldsymbol{p}$
```

Algorithm 6 `findBestColumnAccordingToSurrogate`$(\mathcal{G}, \lambda, \boldsymbol{b}, a)$

```
 1: for all particles $i$ do
 2:     $[\boldsymbol{p}^i, \boldsymbol{v}^i] \leftarrow$ initialize()
 3: end for
 4:
 5: repeat
 6:     for all particles $i$ do
 7:         cost $\leftarrow$ calculateSurrogateCost($\lambda, \boldsymbol{b}, a, \boldsymbol{p}^i$)
 8:         $\boldsymbol{p}^i_{\text{pBest}}$ $\leftarrow$updateParticleBest(cost, $\boldsymbol{p}^i_{\text{pBest}}$)
 9:         $\boldsymbol{p}_{\text{gBest}}$ $\leftarrow$updateSwarmBest(cost, $\boldsymbol{p}_{\text{gBest}}$)
10:     end for
11:
12:     for all particles $i$ do
13:         $\boldsymbol{v}^i \leftarrow$ updateVelocity(pBest$^i$, gBest, $\boldsymbol{p}^i$)
14:         $\boldsymbol{p}^i \leftarrow$ updatePositionUsingGraph($\mathcal{G}, \boldsymbol{p}^i, \boldsymbol{v}^i$)
15:     end for
16: until maxIter
17: return alternativeColumns
```

5.2.5 Modification of an already sampled column

The column found as the shortest path or by the BPSO might correspond to a column that has already been computed. Such a column could therefore not be used to refit the model since the model parameters in the next iteration would remain unchanged. To prevent the algorithm from terminating, the duplicate column was hence modified such that a new column was constructed. By randomly removing one task at a time, until a new column was found, the iterative process could continue.

For some columns, removing tasks randomly still resulted in already sampled columns. In those cases, a completely new column was created by randomly trying to assign the tasks in $\mathcal{T}'_k$. As soon as a new column that had not been sampled previously was generated, the algorithm could continue. The complete modification procedure is presented in Algorithm 7.

Algorithm 7 `modifyColumn`($\boldsymbol{p}$)

```
 1: $\mathcal{T}_p \leftarrow$ getTasksInColumn($\boldsymbol{p}$)
 2: $\mathcal{T}_p \leftarrow$ shuffle($\mathcal{T}_p$)
 3:
 4: for all tasks $t \in \mathcal{T}_p$ do
 5:     $\boldsymbol{p} \leftarrow$ removeTask($\boldsymbol{p}, t$)
 6:     if !isAlreadySampled($\boldsymbol{p}$) then
 7:         break
 8:     end if
 9: end for
10:
11: if !isAlreadySampled then      // Create random column
12:     $\boldsymbol{p} \leftarrow$ createEmptyColumn()
13:     for all tasks $t \in \mathcal{T}'_k$ do
14:         $\boldsymbol{p} \leftarrow$ tryAssign($t$)
15:         if !isAlreadySampled($\boldsymbol{p}$) then
16:             break
17:         end if
18:     end for
19: end if
20: return $\boldsymbol{p}$
```

5.2.6 The complete algorithm

A complete overview of the two algorithms using surrogate modeling is shown in Algorithm 8. Since the two pricing methods only differ in how the candidate points are found, they are presented together, where the flag `useLinearization` is used to determine if the candidate point should be taken as the corresponding shortest path in the graph $\mathcal{G}$ over all tasks in $\mathcal{T}'_k$ using a linear approximation of the surrogate function, or a BPSO should be used to minimize the nonlinear surrogate function.

First, the task selection and the construction of the initial set of columns is performed. The graph over all tasks in $\mathcal{T}'_k$ is set up by checking all pairwise connection in Rave (`setupGraph`). Then, for a fixed number of iterations, a candidate solution is found that is used to re-fit the surrogate model. The candidate solution found

Algorithm 8 `solvePricingUsingSurrogateModeling(useLinearization)`

```
1: $[S, \mathcal{T}_k']$ ← selectTasksAndGetInitialColumns()
2: $\mathcal{G}$ ← setupGraph($\mathcal{T}_k'$)
3: $[P, \Phi]$ ← constructMatrices($S$)
4: alternativeColumns ← [ ]       // Columns with negative reduced cost
5:
6: repeat
7:     $\boldsymbol{p}$ ← empty
8:     $[\lambda, \boldsymbol{b}, a]$ ← solveEquationSystem($P$, $\Phi$)
9:
10:    if useLinearization then
11:        $\boldsymbol{p}$ ← findBestColumnAccordingToLinearSurrogate($\mathcal{G}, \lambda, \boldsymbol{b}, a$)
12:    else
13:        $\boldsymbol{p}$ ← findBestColumnUsingSurrogate($\mathcal{G}, \lambda, \boldsymbol{b}, a$)
14:    end if
15:
16:    if isAlreadySampled($\boldsymbol{p}$) then
17:        $\boldsymbol{p}$ ← modifyColumn($\boldsymbol{p}$)
18:    end if
19:
20:    if isLegal($\boldsymbol{p}$) then
21:        reducedCost ← calculateReducedCost($\boldsymbol{p}$)
22:        if reducedCost $< 0$ then
23:            alternativeColumns ← saveColumnToRMP($\boldsymbol{p}$)
24:        end if
25:    else
26:        reducedCost ← setHighReducedCostForIllegalRoster()
27:    end if
28:
29:    $[P, \Phi]$ ← updateMatrices($\boldsymbol{p}, P, \Phi$)
30:    $S$ ← addColumn($\boldsymbol{p}, S$)
31: until maxIter
32: return alternativeColumns
```

either by solving a shortest path problem or using BPSO, might be illegal since the tasks in the column are only ensured to be pairwise legal. Therefore, a legality check is performed by calling Rave. If the column is legal, the true reduced cost is calculated according to (3.8) after calling Rave; if the reduced cost is negative, the column is added to the set of columns that might enter the RMP. On the other hand, if the column turns out to be illegal, a high cost is set to prevent the column from being re-sampled and similar columns from being computed in the following iteration. Finally, the column is included in S before moving on to the next iteration.

6
Test and results

The alternative pricing methods were implemented in C++ in the existing column generation framework at Jeppesen. Each test was run on a machine with Intel(R) Xeon(R) CPU E5-2667 v2 @ 3.30GHz.

The implemented methods were tested on real data from five different problem instances from five different customers. The number of crew members and the number of unassigned tasks for each test case are listed in Tab. 6.1 together with the median size of the associated pricing problems. The size of a pricing problem is determined by the number of tasks that could be assigned to the corresponding crew member. Since some tasks might not be assignable to certain crew members, due to for example the wrong rank, the sizes of the pricing problems differ within the same test case.

Table 6.1: The approximate number of crew members and tasks in each test case and the median size of each pricing problem, i.e. the number of tasks included.

Test case	**Number of crew**	**Number of tasks**	**Median pricing size**
1	600	4 000	2 400
2	1 000	3 000	1 700
3	2 300	3 500	400
4	1 700	3 500	1 700
5	600	3 000	1 400

The different test cases in Tab. 6.1 are of different sizes and come from different airlines which means that they are also different with respect to rules and regulations. In the first test case the proportion of tasks compared to the size of the crew is much higher than for the other test cases and the median number of tasks that are possible to assign to each crew member is also the highest. This means that the number of tasks assigned in each roster will be higher than for the other test cases. For the third test case, which is the largest instance with respect to the number of crew members and with a proportion of tasks against crew that is significantly smaller, a different type of roster will be generated. Test cases 2 and 4 are more similar in size, although test case 4 is slightly bigger. The last test case is, with respect to size, similar to the first one, however, not as extreme.

The three alternative pricing methods, the method using BPSO directly to the pricing problem; LinSurMod, where a surrogate function was approximated linearly; and SurMod, where the surrogate function was minimized heuristically, were run on each test case separately. The methods were called at the same phase in the column generation process such that their performances could be compared. For each test, the same model parameters were used, and no tuning was performed with respect to single tests. For the BPSO method, ten particles were used and the algorithm was run for 50 iterations before terminating. The transfer function $V_4(v) = \left|\frac{2}{\pi}\arctan\left(\frac{\pi}{2}v\right)\right|$ was used, the velocity was limited to $v_{\max} = 3$, the inertia weight was initially set to $w = 1.3$, having a constant decay of 0.98 and a minimum bound of 0.4, and the constants c_1 and c_2 were both set to 2.

For the two pricing methods using the surrogate modeling approach, LinSurMod and SurMod, the surrogate function was constructed using the thin plate spline RBF $\phi(r) = r^2\log(r)$. This function was chosen since an initial comparison between different RBFs gave favorable results for the thin plate spline RBF. Depending on if a linearization of the surrogate function was made or not, slightly different parameters were chosen. When solving the pricing problem with the linearized version of the function (LinSurMod), the algorithm was run for 450 iterations, which means that 450 columns were investigated. When solving the nonlinear surrogate function using BPSO (SurMod), the S-shaped transfer function $S_2(v) = \frac{1}{1+e^{-v}}$ was used to determine if a task should be assigned or not. The parameters used in the update rule for the velocity (4.3) were the same as for the algorithm using BPSO directly. The method was run for 50 iterations, which means that for this approach only 50 columns were investigated, i.e. evaluated in Rave. In each iteration, the BPSO algorithm was run for 50 iterations, where the best column found in these iterations was evaluated in Rave. This choice was made so that the corresponding run times would be similar to the ones using the linearization, in order to get a fair comparison between the performance of the two methods.

Table 6.2: The average run time in CPU seconds of solving one pricing problem for the three methods investigated.

Test case	**BPSO**	**LinSurMod**	**SurMod**
1	8.5	4.7	9.5
2	3.8	3.7	4.0
3	5.4	4.7	8.0
4	3.1	3.4	3.4
5	1.7	3.4	2.9

The average run time of solving one pricing problem, i.e. trying to find columns with negative reduced costs for a specific crew member given a set of specific dual variable values, for the three different methods are presented in Tab. 6.2. Note that, because of the difficulty of solving these pricing problems, columns with negative reduced costs might not always be found. It is also important to point out that the test cases have different rule and cost structures which will influence the run times. Test cases 1 and 3 take the longest to run for all three methods and since these problems are the largest ones, longer run times are expected. The second method, LinSurMod, has similar run times for all test cases, although with slightly increased values for test cases 1 and 3. The run times fluctuate more for the BPSO method and the SurMod method, and these two methods show similar trends. The first test case possesses the largest pricing problems, which means that columns should be formed out of a larger set of possible tasks and that rosters with more tasks assigned should be generated. For the BPSO method, where no task selection is performed, a larger problem space is searched and this may explain partly the increased run time. Also, if many assignments should be evaluated when updating a particle's position, this would lead to longer run times, which would affect the surrogate modeling approach as well.

Two different performance measures are used to compare the three different models; the ability to find columns with negative reduced cost and the improvement in objective value of the RMP in each iteration. For each test case, the methods were run separately but called at the same phase of the column generation process. Since it can be assumed to be more difficult to find columns with negative reduced cost towards the end of this process, each method was called at an early stage of the column generation framework and then again at a later phase.

6.1 Ability to find columns with negative reduced cost

The first measure of the performance is the methods' ability to find columns with negative reduced cost. This can be measured both from the hit rate, i.e. the percentage of pricing problems for which the method was able to find columns with a negative reduced cost, and from the value of the best reduced costs found by the methods for all pricing problems at the early/later phase of each test case.

To compare the performances, each method was used to solve the exact same set of pricing problems from each test case and each method was called by the existing column generation algorithm at the two different phases of the column generation process. In order for the methods to be compared properly, the found columns with negative reduced cost were not allowed to enter the RMP. Since each method finds different columns, the dual variables in the next iteration would be different. Hence, the subsequent pricing problems would not be equal for the three methods and the corresponding results would not be directly comparable. Therefore, the alternative pricing methods were run simultaneously with the current pricing methods at Jeppesen, but only the columns found by the existing pricing methods were added to the RMP. Between the early and later phase, only the current pricing methods were run.

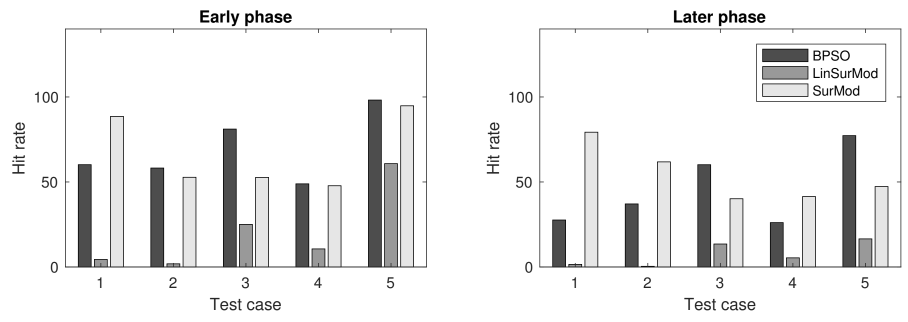
The results for each test case are presented in Tab. 6.3 and illustrated in Fig. 6.1 and the results for the early and later phases are presented separately. In the table, the hit rates as well as the mean of negative reduced costs and the lowest reduced cost found by each method, are presented. The two latter give an indication of the quality of the columns found. The mean of reduced costs was taken using only the reduced costs from the pricing problems for which the method was able to find at least one column with negative reduced cost. For example, for the first test case at the early phase, the BPSO pricing method has a hit rate of 60.17%, so the mean was taken over the best reduced costs found from these 60.17% successful runs. Note that the two measures of reduced cost have been normalized separately for each test case and phase, with respect to the best result found by the three methods, i.e.

$$\text{normalizedResult}_m^{\text{phase}} = \frac{\left|\text{result}_m^{\text{phase}}\right|}{\max_{\mu\in\mathcal{M}}\left\{\left|\text{result}_\mu^{\text{phase}}\right|\right\}}, \quad m \in \mathcal{M}, \tag{6.1}$$

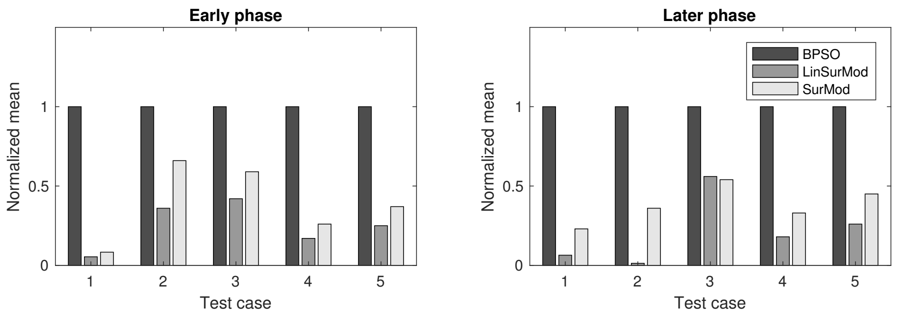
where $\mathcal{M} = \{\text{BPSO, LinSurMod, SurMod}\}$ is the set of the three alternative methods and ”phase” signifies either the early or later phase. This means that the most successful pricing method will have the value 1. The reduced costs found cannot be compared between test cases, since each test case has different cost structure, and therefore, a separate normalization for each test case can be performed without loosing any relevant information.

Table 6.3: The results with respect to reduced cost for the three different pricing methods for the early and later phases of the columns generation process of the five test cases listed in Tab. 6.1.

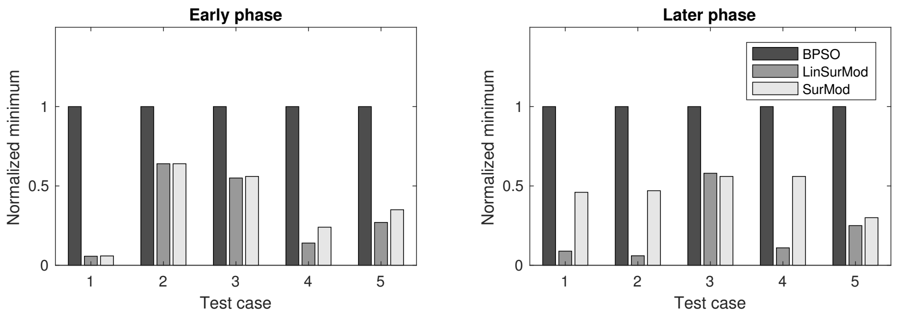
	Early phase			**Later phase**		
Test case	**BPSO**	**LinSurMod**	**SurMod**	**BPSO**	**LinSurMod**	**SurMod**
			Hit rate (%)			
1	60.17	4.41	88.57	27.62	1.50	79.27
2	58.19	1.85	52.75	37.08	0.38	61.82
3	81.11	25.04	52.70	60.15	13.50	40.13
4	48.90	10.62	47.78	26.09	5.37	41.43
5	98.18	60.78	94.81	77.26	16.52	47.30
			Normalized mean of absolute value of reduced costs for successfully solved pricing problems			
1	1	0.054	0.083	1	0.064	0.23
2	1	0.36	0.66	1	0.013	0.36
3	1	0.42	0.59	1	0.56	0.54
4	1	0.17	0.26	1	0.18	0.33
5	1	0.25	0.37	1	0.26	0.45
			Normalized minimum of absolute value of reduced costs for successfully solved pricing problems			
1	1	0.057	0.059	1	0.089	0.46
2	1	0.64	0.64	1	0.060	0.47
3	1	0.55	0.56	1	0.58	0.56
4	1	0.14	0.24	1	0.11	0.56
5	1	0.27	0.35	1	0.25	0.30



(a) The hit rate for each test case given by the three methods.



(b) The normalized mean of absolute values of reduced costs for successfully solved pricing problems for the three methods.



(c) The normalized minimum of absolute values of reduced costs for successfully solved pricing problems for the three methods.

Figure 6.1: The performance results for the three methods corresponding to Tab. 6.3.

The results for the early phase, columns 2–4 in Tab. 6.3, show that the LinSurMod method has the lowest hit rates for all test cases. The hit rates are especially low for test cases 1 and 2, see the left bar plot in Fig. 6.1a. However, the hit rate for test case 5 is much higher than for the other test cases. Since the hit rate is always worse for LinSurMod compared to SurMod, it indicates that the linearization does not preserve the characteristics of the true cost function. Therefore, the shortest path using the linear approximation in LinSurMod does not equal the best column according to the nonlinearized surrogate function.

The hit rates for the BPSO method and the SurMod method are closer. The surrogate modeling approach has higher hit rate for the first test case, but on the other hand, the BPSO method performs better for the third test case. For the remaining three cases, the hit rates are quite similar, but slightly higher for the BPSO.

With respect to the negative reduced costs of the columns found in the early phase by the methods, see the left bar plots in Fig. 6.1b and 6.1c, LinSurMod finds columns with the highest reduced costs. However, compared to the hit rates, the results for the mean and the minimum of negative reduced costs found by LinSurMod and SurMod, are closer. The BPSO method finds columns with the lowest reduced costs for all test cases. Especially for the first test case, both the mean and minimum of reduced costs found by the BPSO, is much lower than for the two other methods. This could either mean that the performance is poor for the two surrogate modeling methods or that the first method is very successful in finding columns with very low negative reduced costs for at least some of the pricing problems considered.

For the hit rates corresponding to the later phase of the test cases, columns 5–7 in Tab. 6.3, SurMod has the highest performance for three of the test cases (1, 2, 4). It is also worth noticing that the hit rates decrease toward the later phase. Since it is less probable to find columns with negative reduced cost towards the end of the column generation process, this is expected. However, an exception is seen for the second test case using SurMod, where the hit rate increases with around nine percentage points. Overall, the decrease in hit rates between the early and later phase seems to be less significant for SurMod compared to the other two methods.

At the later phase, the columns with the lowest reduced cost for all five test cases are found by the BPSO method, just as for the early phase.

6.2 Improvement in objective value of the restricted master problem

The second performance measure is the improvement in objective value of the RMP. The improvement is defined as the difference in objective value for the RMP before and after columns have been generated using one of the alternative methods, divided by the total number of pricing problems the method has been applied to, i.e.,

$$\text{improvement} = \frac{\Delta \text{ objective value}}{\# \text{ pricing problems}}. \tag{6.2}$$

The results are presented in Tab. 6.4 and illustrated in Fig. 6.2. As for the results in Tab. 6.3, the results presented here have been normalized with respect to the best improvement obtained for each test case, separately for each of the two phases (early/later), according to (6.1). The method with the best largest improvement will have the value 1.

For each test case only the columns found by the alternative pricing method, and not the columns found by the existing pricing methods, were allowed to enter the RMP. In this way, the presented improvement in objective value is only affected by

Table 6.4: The average normalized improvement in objective value for the RMP.

	Early phase			**Later phase**		
Test case	**BPSO**	**LinSurMod**	**SurMod**	**BPSO**	**LinSurMod**	**SurMod**
1	0.96	0.013	1	0.22	0.032	1
2	1	0.056	0.45	1	0.0010	0.36
3	1	0.54	0.58	1	0.68	0.80
4	1	0.10	0.85	0.89	0.060	1
5	1	0.080	0.54	1	0.081	0.41

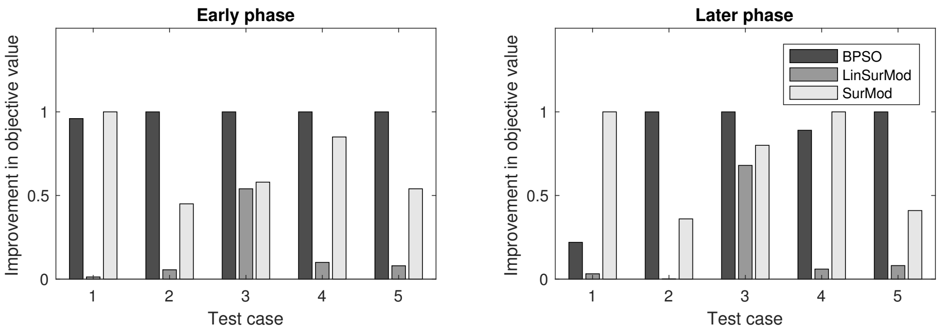


Figure 6.2: The improvement in RMP given by the three methods at the early/later phase.

the alternative method and not influenced by the existing pricing methods. When allowing the generated columns to enter the RMP, different sets of dual variable values of the RMP are obtained and therefore the following pricing problems will not be the same for the three different methods, as was the case for the results presented in Tab. 6.3. In the iterations where the alternative pricing methods were not used, only the existing pricing methods were used to generate and add columns to the RMP.

The pricing methods were called at the same occasion in the column generation process as in the previous tests which means that some comparison between the results presented in Tables 6.3 and 6.4 can be made, for example if a high hit rate has an impact on objective value or if it is important to find columns with as low reduced costs as possible.

For most test cases, the largest improvement with respect to objective value is given by the BPSO method, with exceptions for test cases 1 and 4. This could be expected since this pricing method also performed best with respect to finding columns with negative reduced cost. The columns found had the lowest reduced cost, both with respect to mean and minimum, as well as a relatively high hit rate and is therefore likely to result in a larger improvement. For test case 1, the performance of the BPSO method, is worse compared to the results given by SurMod, especially at the later phase of the process. From Tab. 6.3, it can be seen that the hit rate at this stage is only 27.62% for the BPSO method compared to 79.27% for the SurMod method. This indicates that even though the BPSO method is able to find the columns with the lowest reduced cost, the hit rate is too low and therefore SurMod performs better. However, at the early phase of the column generation process the hit rate using the BPSO method, is relatively high, and together with the fact that columns with much lower reduced costs have been found compared to the other two methods, the performance is much closer to the one given by SurMod.

The SurMod method has a performance close to that of the method using the BPSO method for test case 4 and gives a larger improvement of objective value at the later phase of the test. Similar to test case 1, the hit rate is higher (41.43%) using SurMod compared to the BPSO method (26.09%), and even though the BPSO method finds columns with lower reduced cost, SurMod gives a larger improvement.

The LinSurMod method gives the overall worst improvement in objective value. However, for test case 3 the method performs quite similar to the other two methods. Compared to SurMod the performance is even slightly better at the early phase. From the reduced cost measures in Tab. 6.3, this is the test case for which LinSurMod finds the best columns, or at least finds columns which are more comparable to the ones found by the BPSO method and SurMod. In combination with a hit rate that is relatively good, at least in comparison with the hit rates obtained using LinSurMod for test cases 1,2,4 and 5, this could explain why the performance of LinSurMod is higher.

For test cases 1,2,4 and 5, given the results in Tab. 6.3, it is apparent that LinSurMod struggled to find columns with negative reduced cost. Therefore, it is not surprising that the improvement in objective value for the RMP is small as well. If only a small number of columns can enter the RMP and these columns also have reduced cost close to zero, the improvement in objective value will be small. This can be seen especially for the second test case.

It is also notable that even if the hit rate was quite high in the early phase of test case 5, the improvement in objective value of the resulting RMP is quite low. On the other hand, the hit rate was higher for the BPSO method and the SurModmethod, and these methods were also able to find columns with lower reduced cost, and since the improvement of objective value is compared to the best result obtained, it is not surprising that the improvement was higher using the BPSO method and the SurMod method.

7 Conclusion and discussion

In this thesis, three alternative pricing methods were implemented to solve the pricing problem in column generation for the airline crew rostering problem, which consists of finding columns corresponding to rosters with negative reduced costs. A pricing problem is solved for one crew member at a time, using optimal dual variable values from the current restricted master problem (RMP). In order to be applicable within any airline, the pricing methods need to be independent of airline-specific rules; hence, the pricing problem is regarded as a black-box optimization problem.

The first pricing method is a binary particle swarm optimization (BPSO) algorithm that performs a search over all tasks that can be assigned to the crew member for which the pricing problem is solved. The second and third pricing methods, LinSurMod and SurMod, use a surrogate modeling approach where a surrogate function is constructed using radial basis functions. The surrogate function is used instead of the true objective function to search for columns. In the first approach, the surrogate function is linearly approximated such that linear costs with respect to each task can be derived. Columns are then found by solving a shortest path problem, where each node represents one of the assignable tasks, and the edge costs are taken from the linearized surrogate function. In the second approach, the nonlinear surrogate function is minimized heuristically using a BPSO algorithm to investigate the benefits from not further approximating the nonlinear surrogate function. The found solutions to the surrogate function are then evaluated using the true cost function. For the surrogate modeling methods, a subset of tasks is selected; and hence a more local search is conducted since the methods are restricted to only finding columns covering the selected tasks.

The three methods were evaluated on five real-world test cases at two different phases; an early phase and a later phase. The results were quite consistent. The pricing method using a BPSO to solve the pricing problem over all assignable tasks had the best overall performance. Although the run times fluctuated, mostly due to the size of the pricing problems solved, the algorithm performed quite well independently of the size of each pricing problem. Compared to the other methods, the BPSO method found the columns with lowest reduced cost for all the test cases and also had a relatively high hit rate. The BPSO also gave rise to the largest improvement in objective value of the RMP for most test cases.

With respect to hit rate, the surrogate method that uses the nonlinear surrogate function to search for new columns, SurMod, provided results closer to the BPSO method and even performed better for a few of the test cases. For most of the test instances where the hit rate was higher for SurMod compared to the BPSO method, the improvement in objective value for the RMP was also higher. This could indicate that, although SurMod finds columns with higher reduced cost, it is more important to have a high hit rate.

The hit rates decreased towards the later phase of the column generation process. However, the performance decrease was less significant for SurMod. This might be explained by the fact that when the method is called again towards the later phase of the test, more columns are included in the RMP, which means that the surrogate function is based on more columns that had been found by solving previous pricing problems.

The SurMod method performed much better than the method that performs a linear approximation of the same surrogate function, LinSurMod. This indicates that the linearization does not preserve the ability to rank columns in the correct order with respect to reduced cost and therefore failed to find columns with low reduced cost. This means that the linear approximation of the nonlinear surrogate function correspond to neither equal shortest paths nor equal best columns.

The main benefit of fitting a surrogate model to the cost function is that an explicit expression is obtained which can be used to search for new columns using, for example, gradient information. Since a BPSO algorithm was used, the real benefits of using a surrogate function was not really utilized. However, since the attempt to solve the nonlinear function was more successful, it indicates that the nonlinear function is able to capture at least some aspects of the true cost function. Therefore, as a future development, it is interesting to investigate more efficient solution methods for finding the global optimum of the surrogate function, for example using a mixed integer nonlinear problem solver such as Bonmin (Basic open-source nonlinear mixed integer programming) [42] and the nonlinear solver SLP offered by FICO®Xpress Optimization [43].

In the BPSO pricing method, all tasks that can be assigned to the crew member were allowed to be assigned. However, for the two surrogate modeling methods, LinSurMod and SurMod, a selection of tasks was performed; see Section 5.2.1. A possible explanation for the fact that the BPSO method found columns with lower reduced cost is that these columns contain tasks that are not allowed in the surrogate models. By performing a selection of tasks based on the columns in the RMP, the methods will be restricted to columns that have possibly already been investigated. On the other hand, the number of columns that can be formed from even a small set of tasks is extremely large, so even in this case, many new columns can be found. The absolute best columns may, however, have been excluded.

To understand the impact that the selection of tasks has on the algorithm, a more extensive evaluation have to be performed. For example, the BPSO method could be run using the same task selection as the one performed for LinSurMod and SurMod, and if columns with higher reduced costs are found, this would indicate that the selection is too restrictive. If the reduced costs are not influenced, then by applying the BPSO method on a smaller subset of tasks, the run times would be heavily reduced.

For the BPSO method, one complication was that because of the sheer size of the problem, the algorithm tries to assign too many tasks. By limiting the number of new tasks that the method may try to assign, the run times would be reduced. Also, it could be interesting to initialize the velocities in the BPSO algorithm differently for each tasks and not uniformly random. Since the tasks that were selected for the two surrogate modeling methods still allowed for some columns with negative reduced cost to be found, these tasks may be initialized with a slightly larger probability of getting assigned than tasks that are not part of the selection. This would guide the search towards tasks that had been assigned to previously found columns without excluding any tasks.

For the methods using the surrogate function, some computational time had to be spent on solving the linear system of equations to obtain the parameters to the surrogate function. This was done by performing an LU factorization. Since such a system of equation should be solved in each iteration, where the equation system only changes slightly, a method that can take advantage of the previous factorization would provide a faster solution method.

The pricing run times for the developed methods were too high to be competitive with the current pricing methods at Jeppesen. However, the focus of this thesis has not been to compare the developed alternative methods with the ones currently used and the only time limitation was that the methods should be able to run within a reasonable amount of time. Therefore, little efforts with respect to reducing the pricing run times or implementing the methods in the most effective way have been made. In the future, efforts must be made to reduce the computational times such that the true potential of each method can be evaluated. However, the results with respect to reduced cost and improvement in objective value for the RMP looked promising, but an in-depth comparison between the current pricing methods and the alternative methods needs to be performed.

To conclude, all three alternative pricing methods were able to find columns with negative reduced costs, although with different success rate. Especially, the BPSO pricing method and the nonlinear surrogate modeling approach SurMod, show promising results. However, improvements with respect to run times as well as parameter tuning are needed together with more extensive evaluations.

Bibliography

[1] *State of the Industry and Global Economic Outlook*. https://www.iata.org/pressroom/media-kit/Documents/State-of-the-industry-and-global-economic-outlook.pdf. Accessed: 2018-05-13.

[2] Niklas Kohl and Stefan E. Karisch. “Airline crew rostering: Problem types, modeling, and optimization”. In: *Annals of Operations Research* 127.1 (2004), pp. 223–257. DOI: 10.1023/B:ANOR.0000019091.54417.ca.

[3] Jacques Desrosiers and Marco E. Lübbecke. “A Primer in Column Generation”. In: *Column Generation*. Ed. by G. Desaulniers, J. Desrosiers, and M. Solomon. Boston, MA, USA: Springer, 2005, pp. 1–32. DOI: 10.1007/0-387-25486-2_1.

[4] Panta Lučic and Dušan Teodorovic. “Simulated annealing for the multi-objective aircrew rostering problem”. In: *Transportation Research Part A: Policy and Practice* 33.1 (1999), pp. 19–45. DOI: https://doi.org/10.1016/S0965-8564(98)00021-4.

[5] R. Hadianti, K. Novianingsih, S. Uttunggadewa, K.A. Sidarto, N. Sumarti, and E. Soewono. “Optimization model for an airline crew rostering problem: Case of Garuda Indonesia”. In: *Journal of Mathematical and Fundamental Sciences* 43.3 (2013), pp. 218–234. DOI: "10.5614/j.math.fund.sci.2013.45.3.2".

[6] Michel Gamache, François Soumis, Gérald Marquis, and Jacques Desrosiers. “A column generation approach for large-scale aircrew rostering problems”. In: *Operations Research* 47.2 (1999), pp. 247–263.

[7] Michel Gamache and François Soumis. “A Method for Optimally Solving the Rostering Problem”. In: *Operations Research in the Airline Industry*. Ed. by Gang Yu. Boston, MA: Springer US, 1998, pp. 124–157. DOI: 10.1007/978-1-4615-5501-8_5.

[8] Silke Jütte, Daniel Müller, and Ulrich W. Thonemann. “Optimizing railway crew schedules with fairness preferences”. In: *Journal of Scheduling* 20.1 (2017), pp. 43–55. DOI: 10.1007/s10951-016-0499-4.

[9] Silke Jütte, Marc Albers, Ulrich W. Thonemann, and Knut Haase. “Optimizing Railway Crew Scheduling at DB Schenker.” In: *Interfaces* 41.2 (2011), pp. 109–122.

[10] Jonathan F. Bard and Hadi W. Purnomo. “Preference scheduling for nurses using column generation”. In: *European Journal of Operational Research* 164.2 (2005), pp. 510–534. DOI: https://doi.org/10.1016/j.ejor.2003.06.046.

[11] Broos Maenhout and Mario Vanhoucke. “A hybrid scatter search heuristic for personalized crew rostering in the airline industry”. In: *European Journal of Operational Research* 206.1 (2010), pp. 155–167. DOI: https://doi.org/10.1016/j.ejor.2010.01.040.

[12] Nadia Souai and Jacques Teghem. “Genetic algorithm based approach for the integrated airline crew-pairing and rostering problem”. In: *European Journal of Operational Research* 199.3 (2009), pp. 674–683. DOI: https://doi.org/10.1016/j.ejor.2007.10.065.

[13] R.S. Garfinkel and G.L. Nemhauser. “THE SET-PARTITIONING PROBLEM: SET COVERING WITH EQUALITY CONSTRAINTS.” In: *Operations Research* 17.5 (1969), pp. 848–856.

[14] Kurt Mehlhorn and Mark Ziegelmann. “Resource Constrained Shortest Paths”. In: *Algorithms — ESA 2000*. Ed. by M. Paterson. Springer Berlin, 2000, pp. 326–337.

[15] Torsten Fahle, Ulrich Junker, Stefan E. Karisch, Niklas Kohl, Meinolf Sellmann, and Bo Vaaben. “Constraint programming based column generation for crew assignment”. In: *Journal of Heuristics* 8.1 (2002), pp. 59–81. DOI: 10.1023/A:1013613701606.

[16] Meinolf Sellmann, Kyriakos Zervoudakis, Panagiotis Stamatopoulos, and Torsten Fahle. “Crew Assignment via Constraint Programming: Integrating Column Generation and Heuristic Tree Search”. In: *Annals of Operations Research* 115.1 (2002), pp. 207–225. DOI: 10.1023/A:1021105422248.

[17] Steven A Gabriel and David Bernstein. “Nonadditive Shortest Paths: Subproblems in Multi-Agent Competitive Network Models”. In: *Computational & Mathematical Organization Theory* 6.1 (2000), pp. 29–45. DOI: 10.1023/A:1009621108971.

[18] Peng Chen and Yu (Marco) Nie. “Bicriterion shortest path problem with a general nonadditive cost”. In: *Transportation Research Part B: Methodological* 57 (2013), pp. 419–435. DOI: https://doi.org/10.1016/j.trb.2013.05.008.

[19] Florence Massen, Yves Deville, and Pascal Van Hentenryck. “Pheromone-Based Heuristic Column Generation for Vehicle Routing Problems with Black Box Feasibility”. In: *Integration of AI and OR Techniques in Contraint Pro-*

gramming for Combinatorial Optimzation Problems. Ed. by N. Beldiceanu, N. Jussien, and E. Pinson. Springer Berlin, 2012, pp. 260–274.

[20] A. R. Conn, Katya Scheinberg, Luis N. Vicente, Books24x7 (e-book collection), and Inc Books24x7. *Introduction to derivative-free optimization.* English. Vol. 8;8.; Philadelphia: Society for Industrial and Applied Mathematics/Mathematical Programming Society, 2009.

[21] Nathan Bell and B John Oommen. "A novel abstraction for swarm intelligence: particle field optimization". In: *Autonomous Agents and Multi-Agent Systems* 31.2 (2017), pp. 362–385. DOI: 10.1007/s10458-016-9350-8.

[22] Hirotaka Nakayama and M. Arakawa. "Using support vector machines in optimization for black-box objective functions". In: *Neural Networks, 2003.* 2 (2003), pp. 1–6. DOI: 10.1109/IJCNN.2003.1223941.

[23] Mattias Björkman and Kenneth Holmström. "Global Optimization of Costly Nonconvex Functions Using Radial Basis Functions". In: *Optimization and Engineering* 1.4 (2000), pp. 373–397. DOI: 10.1023/A:1011584207202.

[24] Leon S. Lasdon. *Optimization Theory for Large Systems.* Mineola, New York: Dover Publications, 1970.

[25] Ashok D. Belegundu and Tirupathi R. Chandrupatla. *9.3 Branch and Bound Algorithm for Mixed Integers (LP-Based).* 2011.

[26] Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, and Pamela H. Vance. "Branch-and-Price: Column Generation for Solving Huge Integer Programs". In: *Operations Research* 46.3 (1998), pp. 316–329.

[27] James Kennedy and Russell C. Eberhart. "Particle swarm optimization". In: *Proceedings of the 1995 IEEE International Conference on Neural Networks.* Vol. 4. Perth, Australia, IEEE Service Center, Piscataway, NJ, 1995, pp. 1942–1948.

[28] Melanie Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, 1998.

[29] Rania Hassan, Babak Cohanim, Olivier de Weck, and Gerhard Venter. "A Comparison of Particle Swarm Optimization and the Genetic Algorithm". In: *46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference.* Reston, Virigina: American Institute of Aeronautics and Astronautics, 2005. DOI: 10.2514/6.2005-1897.

[30] James Kennedy and Russell C. Eberhart. "A discrete binary version of the particle swarm algorithm". In: *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation.* Vol. 5. IEEE, pp. 4104–4108. DOI: 10.1109/ICSMC.1997.637339.

[31] Mattias Wahde. *Biologically inspired optimization methods: an introduction.* English. Southampton: WIT Press, 2008.

[32] Seyedali Mirjalili and Andrew Lewis. “S-shaped versus V-shaped transfer functions for binary Particle Swarm Optimization”. In: *Swarm and Evolutionary Computation* 9 (2013), pp. 1–14. DOI: https://doi.org/10.1016/j.swevo.2012.09.002.

[33] Marc Pirlot. “General local search methods”. In: *European Journal of Operational Research* 92.3 (1996), pp. 493–511. DOI: https://doi.org/10.1016/0377-2217(96)00007-0.

[34] Satyajith Amaran, Nikolaos V. Sahinidis, Bikram Sharda, and Scott J Bury. “Simulation optimization: a review of algorithms and applications”. In: *Annals of Operations Research* 240.1 (2016), pp. 351–380. DOI: 10.1007/s10479-015-2019-x.

[35] Ky Khac Vu, Claudia D’Ambrosio, Youssef Hamadi, and Leo Liberti. “Surrogate-based methods for black-box optimization”. In: *International Transactions in Operational Research* 24.3 (2017), pp. 393–424. DOI: 10.1111/itor.12292.

[36] Humberto Rocha. “On the selection of the most adequate radial basis function”. In: *Applied Mathematical Modelling* 33.3 (2009), pp. 1573–1583. DOI: 10.1016/j.apm.2008.02.008.

[37] Åke Björck. “Direct Methods for Linear Systems”. In: *Numerical Methods in Matrix Computations.* Cham: Springer International Publishing, 2015, pp. 1–209. DOI: 10.1007/978-3-319-05089-8_1.

[38] Kenneth Holmström, Nils-Hassan Quttineh, and Marcus M. Edvall. “An adaptive radial basis algorithm (ARBF) for expensive black-box mixed-integer constrained global optimization”. In: *Optimization and Engineering* 9.4 (2008), pp. 311–339. DOI: 10.1007/s11081-008-9037-3.

[39] Juliane Müller, Christine A. Shoemaker, and Robert Piché. “SO-I: a surrogate model algorithm for expensive nonlinear integer programming problems including global optimization applications”. In: *Journal of Global Optimization* 59.4 (2014), pp. 865–889. DOI: 10.1007/s10898-013-0101-y.

[40] Gaël Guennebaud and Benoît Jacob. *Eigen v3.* http://eigen.tuxfamily.org. Accessed: 2018-05-09.

[41] Jørgen Bang-Jensen and Gregory Z. Gutin. “Distances”. In: *Digraphs: Theory, Algorithms and Applications.* London: Springer London, 2009, pp. 87–126. DOI: 10.1007/978-1-84800-998-1_3.

[42] *Bonmin.* https://projects.coin-or.org/Bonmin. Accessed: 2018-05-09.

[43] *FICO®Xpress Optimization*. `http://www.fico.com/en/products/fico-xpress-optimization`. Accessed: 2018-05-09.

A

Appendix 1: Converting the Euclidean norm into equivalent arc costs

To find a candidate solution using the linearized surrogate function in (5.5), the squared two-norm must be transformed into edge costs in a network such that a shortest path can be found. In the graph representation of the problem of forming rosters, the nodes will be the tasks that can be assigned to the roster and an edge will correspond to connecting two tasks.

Let $\mathcal{N}$ be the set of nodes and $\mathcal{E}$ be the set of edges in the graph, where $ij \in \mathcal{E}$ denotes the edge between nodes $i \in \mathcal{N}$ and $j \in \mathcal{N}$. The two columns, $\boldsymbol{p}$ and $\tilde{\boldsymbol{p}}$, will each represent a path in the network. Let $\mathcal{N}_p$ and $\mathcal{N}_{\tilde{p}}$ be the subset of nodes (tasks) that belong to $\boldsymbol{p}$ and $\tilde{\boldsymbol{p}}$, respectively. Analogously, let $\mathcal{E}_p$ and $\mathcal{E}_{\tilde{p}}$ be the set of edges in $\boldsymbol{p}$ and $\tilde{\boldsymbol{p}}$, respectively. Let $\delta^{(ij)}_{\mathcal{E}_p}$ equal 1 if the edge between node i and j belongs to $\boldsymbol{p}$ and 0 otherwise. Further, let $\delta^{i}_{\mathcal{N}_p} \in \{0, 1\}$ equal 1 if node i belongs to $\boldsymbol{p}$ and 0 otherwise, i.e.,

$$\delta^{ij}_{\mathcal{E}_p} = \begin{cases} 1 \text{ if } ij \in \mathcal{E}_p, \\ 0 \text{ otherwise,} \end{cases} \quad \text{and} \quad \delta^{i}_{\mathcal{N}_p} = \begin{cases} 1 \text{ if } i \in \mathcal{N}_p, \\ 0 \text{ otherwise.} \end{cases} \tag{A.1}$$

The squared two-norm of $(\boldsymbol{p} - \tilde{\boldsymbol{p}})$ can be written as

$$||\boldsymbol{p} - \tilde{\boldsymbol{p}}||_2^2 = \sum_{i \in \mathcal{N}} \left(\delta^{i}_{\mathcal{N}_p} - \delta^{i}_{\mathcal{N}_{\tilde{p}}}\right)^2 = \sum_{i \in \mathcal{N}} \left[(\delta^{i}_{\mathcal{N}_p})^2 - 2\delta^{i}_{\mathcal{N}_p}\delta^{i}_{\mathcal{N}_{\tilde{p}}} + (\delta^{i}_{\mathcal{N}_{\tilde{p}}})^2\right]. \tag{A.2}$$

The right hand side of (A.2) can be simplified by first noting that

$$\sum_{i \in \mathcal{N}} (\delta^{i}_{\mathcal{N}_p})^2 = \sum_{i \in \mathcal{N}} (\delta^{i}_{\mathcal{N}_p}) = |\mathcal{N}_p| = |\mathcal{E}_p| + 1 = \sum_{ij \in \mathcal{E}} \delta^{ij}_{\mathcal{E}_p} + 1, \tag{A.3}$$

where $|\mathcal{N}_p|$ and $|\mathcal{E}_p|$ denote the number of nodes and edges in $\boldsymbol{p}$, respectively. The negative of the second term in (A.2) equals the number of nodes that belong to both path $\boldsymbol{p}$ and $\tilde{\boldsymbol{p}}$, i.e., all nodes $i \in \mathcal{N}_p \cap \mathcal{N}_{\tilde{p}}$. This can be rewritten as a sum over all arcs $ij \in \mathcal{E}$,

$$2 \sum_{i \in \mathcal{N}} \delta^{i}_{\mathcal{N}_p}\delta^{i}_{\mathcal{N}_{\tilde{p}}} = 2\left[\sum_{ij \in \mathcal{E}} \delta^{ij}_{\mathcal{E}_p}\delta^{j}_{\mathcal{N}_{\tilde{p}}} + 1\right], \tag{A.4}$$

i.e. the number of arcs in $\boldsymbol{p}$ that end in a node that belongs to $\tilde{\boldsymbol{p}}$ plus 1 since both paths start at the same node.

Therefore, by inserting (A.3) and (A.4) into (A.2), the following is obtained

$$||\boldsymbol{p}-\tilde{\boldsymbol{p}}||_2^2 = |\mathcal{N}_{\tilde{p}}| - 1 + \sum_{ij\in\mathcal{E}} \delta_{\mathcal{E}_p}^{ij}(1-2\delta_{\mathcal{N}_{\tilde{p}}}^{j}). \tag{A.5}$$

This shows that the expression for the squared two-norm can be divided into a constant cost, $|\mathcal{N}_{\tilde{p}}|-1$, depending only on the number of nodes in $\tilde{\boldsymbol{p}}$ and a cost over the edges in the network depending on if the node to be visited is in $\tilde{\boldsymbol{p}}$.

Assume that $\boldsymbol{p}$ is an arbitrary path in the network, i.e., an arbitrary column, and that $\tilde{\boldsymbol{p}} = \boldsymbol{p}_l$, where $l \in \{1,2,\dots,n\}$, is an already sampled column in the set of sampled columns S. The complete linear approximation of the surrogate function in (5.5) becomes

$$\begin{aligned}\bar{s}(\boldsymbol{p}) &= \sum_{l=1}^{n}\lambda_l\left[|\mathcal{N}_{p_l}| - 1 + \sum_{ij\in\mathcal{E}_p}\delta_{\mathcal{E}_p}^{ij}(1-2\delta_{\mathcal{N}_{p_l}}^{j})\right] + \boldsymbol{\beta}^T\boldsymbol{p} + \alpha \\ &= \sum_{ij\in\mathcal{E}_p}\delta_{\mathcal{E}_p}^{ij}\underbrace{\left[\sum_{l=1}^{n}\lambda_l(1-2\delta_{\mathcal{N}_{p_l}}^{j}) + \beta_j\right]}_{=:\, b_j^{\mathrm{NL}}} + \underbrace{\alpha + \sum_{l=1}^{n}\lambda_l(|\mathcal{N}_{p_l}|-1)}_{=:\, \alpha^{\mathrm{NL}}}.\end{aligned} \tag{A.6}$$

Because of the term $\delta_{\mathcal{E}_p}^{ij}$, the sum over the edges in $\mathcal{E}_p$ can be extended to the sum over all edges in $\mathcal{E}$. Hence, the complete expression for the linearized surrogate model as a function over the edges is given by

$$\bar{s}(\boldsymbol{p}) = \sum_{e_{i,j}\in\mathcal{E}}\delta_{\mathcal{E}_p}^{e_{i,j}} b_j^{\mathrm{NL}} + \alpha^{\mathrm{NL}}, \tag{A.7}$$

where the edge costs are given by $b_j^{\mathrm{NL}} = \sum_{l=1}^{n}\lambda_l(1-2\delta_{\mathcal{N}_{p_l}}^{j}) + \beta_j$, for all edges $e_{i,j}\in\mathcal{E}$ between the two nodes $i,j\in\mathcal{N}$, and the constant cost $\alpha^{\mathrm{NL}} = \alpha + \sum_{l=1}^{n}\lambda_l(|\mathcal{N}_{p_l}|-1)$.

Using this linear approximation of the surrogate function, where each node represents a task t, the constant cost $\hat{\alpha}$ and the complete linear cost $\hat{b}_t$ of each task $t\in\mathcal{T}_k'$ are given by

$$\hat{\alpha} = \alpha^{\mathrm{NL}} + \alpha, \qquad \text{and} \qquad \hat{b}_t = b_t^{\mathrm{NL}} + b_t. \tag{A.8}$$