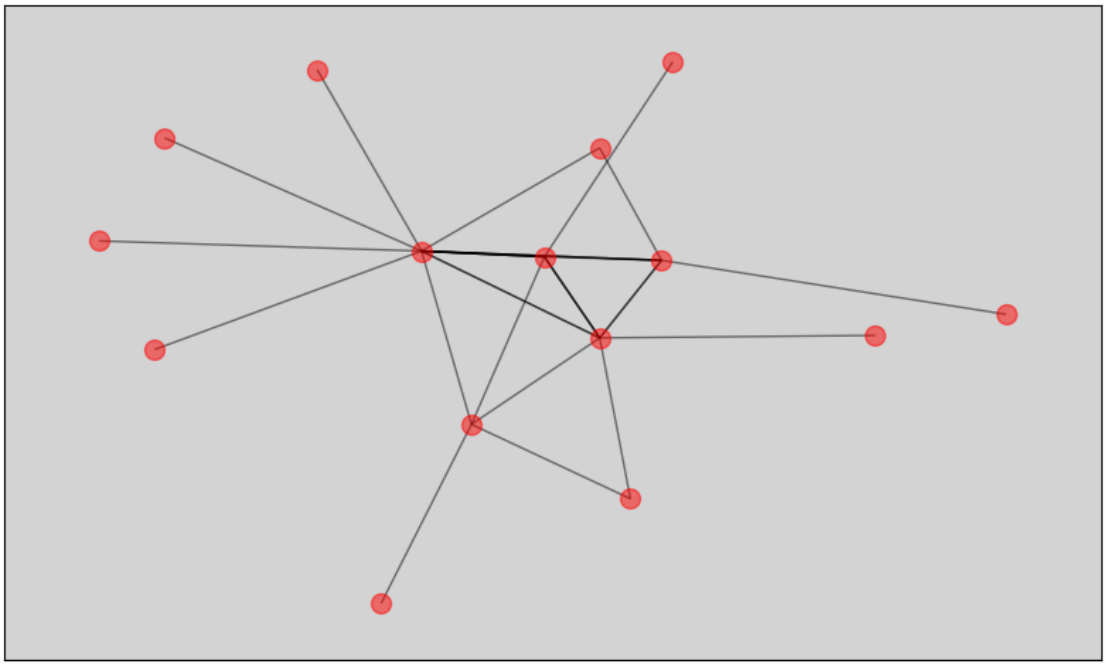


Machine Learning-Enhanced Column Generation for the Crew Pairing Problem

Leveraging Gated Recurrent Units to Generate the Initial Restricted Master Problem

Master's thesis in Engineering Mathematics and Computational Science

Wilmer Eriksson

DEPARTMENT OF MATHEMATICAL SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

Master's thesis 2024

Machine Learning-Enhanced Column Generation for the Crew Pairing Problem

Leveraging Gated Recurrent Units to Generate the Initial Restricted Master Problem

WILMER ERIKSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2024

Machine Learning-Enhanced Column Generation for the Crew Pairing Problem
Leveraging Gated Recurrent Units to Generate the Initial Restricted Master Problem

Wilmer Eriksson, 2024.

Supervisors: Divya Grover, Boeing Gothenburg
Johan Jonasson, Department of Mathematical Sciences
Examiner: Johan Jonasson, Department of Mathematical Sciences

Master's Thesis 2024
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualization of a Flight Network using NetworkX displaying interconnected flight paths.

Printed by Chalmers Reproservice
Gothenburg, Sweden 2024

Machine Learning-Enhanced Column Generation for the Crew Pairing Problem
Leveraging Gated Recurrent Units to Generate the Initial Restricted Master Problem

Wilmer Eriksson
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

The crew scheduling problem is a critical challenge for airlines as crew costs represent the second largest operating cost. It consists of determining the optimal assignment of crew members to each flight leg in an airline's schedule. The problem is broken into two separate problems that are solved sequentially. This research focuses on the initial problem, referred to as the crew pairing problem, where sequences of flight legs, known as pairings, are created. Traditionally, the crew pairing problem is solved by Integer Column Generation frameworks such as the branch-and-price algorithm.

This work proposes the integration of Gated Recurrent Units (GRU) into the state-of-the-art industry algorithms used to solve the crew pairing problem. This method, named the GRU-Enhanced Pairing Initializer (GEPI), generates an initial potential partial solution using supervised learning before starting the traditional column generation process. For large-scale optimization problems, a fundamental trade-off between optimality and computational time exists. Sometimes, problems are not solved to complete optimality; instead, satisfactory or near-optimal solutions are found. Therefore, by improving the starting point of the algorithm, it should be possible to find better solutions.

The results shows that GEPI can generate high-quality pairings that are used in the final solution for 59 out of the 192 test cases. These GEPI-generated pairings reduced the flight schedule cost in 34 test cases, while two cases saw a cost increase. The introduction of GEPI in the optimization process led to an increase of execution time for 37 test cases. These outcomes suggest that there is some potential of improving the starting point of the column generation process using a neural network. However, the decrease in scheduling cost appears to be associated with longer execution times. To gain a more comprehensive understanding of GEPI's potential and limitations, further tests and analysis are needed.

Keywords: Column Generation, Crew Pairing Problem, Gated Recurrent Units, Machine Learning

Acknowledgements

First, I would like to thank my supervisor, Divya Grover, and the rest of the pairing optimization team at Boeing for their guidance during my research. Your insights were instrumental in my work.

I would also like to thank my academic supervisor, Johan Jonasson, for his valuable feedback and encouragement throughout this process.

Wilmer Eriksson, Gothenburg, September 2024

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

CG	Column Generation
EOS	End-Of-Sequence
GEPI	GRU-Enhanced Pairing Initializer
GRU	Gated Recurrent Units
ILP	Integer Linear Program
LP	Linear Program
LSTM	Long Short-Term Memory
MP	Master Problem
NLP	Natural Language Processing
PCA	Principal Component Analysis
RMP	Restricted Master Problem
RNN	Recurrent Neural Network
SVD	Singular Value Decomposition

Contents

1
Introduction

This thesis was developed in collaboration with Boeing's pairing optimization team in Gothenburg, Sweden. This chapter introduces airline scheduling and establishes a definition of the problem at hand.

1.1 Background

Airline schedule optimization is a complex task that is an important indicator of an airline's competitive position [2]. One of the optimization problems in airline scheduling is the crew scheduling problem, which consists of determining the cost-minimizing assignments of pilots and cabin crew, to each flight leg in the airline's schedule. This is a commonly addressed problem, as crew costs represent the second largest operating costs of an airline, with only fuel costs being larger. The crew scheduling problem is often, due to tractability issues, broken down into two stages. First, the crew pairing problem is solved, resulting in sequences of flight legs called pairings. Subsequently, the crew rostering problem is solved, where specific crew members are assigned to the generated pairings.

The crew pairing problem is one of the most challenging problems in airline planning, owing to the many potential pairings, easily in the billions, and the multitude of rules and regulations that need to be followed. The objective of the crew pairing problem is to generate the cost-minimizing set of pairings, ensuring that all flights in the airline's schedule are covered, while following union, government, and airline rules. These rules often include rest periods, minimum layover time, and ensuring that the crew base of the first and last legs in a pairing is the same [8]. One common way of solving the crew pairing problem is by modelling it as a set partitioning problem, which in turn can be formulated as an Integer Linear Program (ILP). Due to its very large-scale nature, the ILP is often solved by approximation algorithms using column generation (CG), to find near-optimal solutions.

1.2 Problem Definition

Boeing's crew pairing optimizer uses a shortest path algorithm to decide which columns to include during column generation, in what's called the pricing problem. Even though the shortest path is a simple algorithm, a significant amount of CPU

time is consumed due to the large size of the network. Another limitation of the current approach is that the pairings generated during the early rounds of column generation often are very simplistic and primarily focus on creating an initial feasible solution. Given that the initial pairings focus on feasibility rather than optimality, they may not contribute to the final optimal solution.

This thesis aims to propose and investigate a machine learning approach to generate faster and/or lower-cost solutions for the crew pairing problem. Specifically, this research will explore the integration of a neural network in the early stage of the optimization process to determine the initial selection of columns prior to starting the traditional column generation procedure. By using a machine learning-based approach for pairing generation, the optimization process could possibly be enhanced by immediately producing higher-quality pairings that contribute, or guide the optimizer, to the final solution.

1.2.1 Research Question

To enhance column generation for the crew pairing problem, this thesis aims to answer the following research question and its sub-questions:

- Can neural networks generate an initial set of pairings that improve the efficiency and quality of solutions in the traditional column generation procedure for the crew pairing problem?
  - How can flight data be represented to leverage a neural network?
  - Can a neural network generalize between data from different airlines?
  - What are the impacts on the column generation process of using a machine learning-based approach for the initial set of pairings?

  .

.

2
Theory

This chapter covers the theoretical foundations and methodologies relevant to this thesis. It begins with key terminology and definitions, followed by an overview of airline schedule optimization and the crew pairing problem. The chapter then covers how to model the crew pairing problem as an ILP, including solution approaches and the dual problem in linear programming. It also dives into the column generation process and the integration of machine learning into these methods. Finally, the chapter covers machine learning techniques used for generating pairings.

2.1 Key Terminology and Definitions in Airline Scheduling

This section provides definitions for key terms used throughout this thesis, specifically in the context of airline scheduling.

Flight Leg

A flight leg refers to a single segment of a trip between two specific airports without any intermediate stops. For example, a direct flight between Gothenburg and Stockholm constitutes a flight leg.

Home Base

The home base is the designated airport a crew member starts and ends their duty period and is the core location for their work assignments.

Deadhead

Deadheading refers to the transportation of a crew member to their next assigned duty location as a non-revenue passenger on an aircraft, which typically is unwanted.

Layover

An overnight rest for the crew between two flight legs.

Pairing and Trip

A pairing or a trip refers to a sequence of connectable flight legs, which starts and ends at the same home base. If the pairing does not end at the same home base, it

is referred to as an incomplete pairing. A pairing typically spans from one to five days and can include layovers and deadheading.

2.2 Airline Schedule Optimization

Airline scheduling involves optimizing the future schedules for aircraft and crews. The goal is to generate profit-maximizing schedules that are aligned with a large set of operational, strategic, and marketing goals. This consists of different complex problems that often are solved sequentially.

First, the *schedule design problem* is solved, detailing the specific flight legs to be operated and a frequency plan designating the days on which each flight leg is scheduled [2]. Subsequently, the *fleet assignment problem* determines the most profitable assignment of aircraft types, for example Boeing 737, to flight legs within the airline's network. This often involves trying to match passenger demand with the number of seats in the specific aircraft types. The next step in the process is to solve the *maintenance routing problem*. The maintenance routing problem consists of assigning each flight leg a specific aircraft of the type decided in the fleet assignment problem. One must also ensure that each individual aircraft is assigned to a sequence of flight legs that allows the aircraft to undergo periodic maintenance checks. Finally, the *crew scheduling problem* is solved, where the objective is to determine the cost-minimizing assignment of pilots and cabin crew to each flight leg in the schedule. A crew schedule includes the sequence of flights and other activities, such as training and vacation leave, that a crew member will perform over a specified period. The optimization of crew schedules is often broken into two stages, due to the complexity of the problem. This thesis focuses on the crew pairing problem, see Section 2.3, where a set of pairings is generated. Specific crew members are assigned to the generated pairings during the last stage of the scheduling process, called the crew rostering problem. An overview of the airline scheduling process can be seen in figure 2.1.

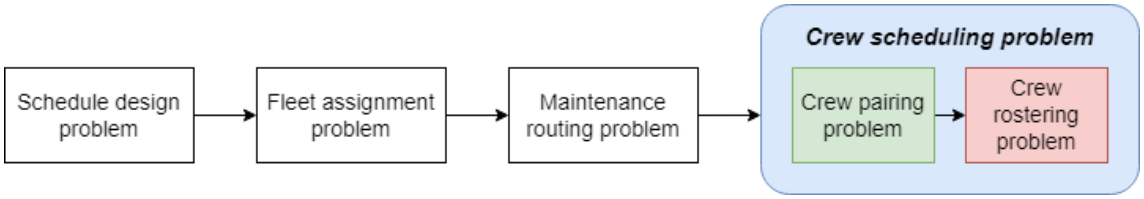


Figure 2.1: An overview of the airline scheduling process.

2.3 Crew Pairing Problem

The input to the crew pairing problem is an airline's timetable, also referred to as the schedule, which consists of a set of flight legs that need to be covered. The set

may also include deadhead flights, which are not required to be covered but can be used to transport crew members between airports. The objective of the crew pairing problem is to generate the minimum-cost set of legal pairings that covers all the required flights in an airline's schedule. A legal pairing can be defined as:

- Starts and ends at the same home base.
- Sequentially ordered both in time and space; that is, the destination airport of an arriving flight must be the same as the departure airport of the following flight in the pairing.
- Follows rules and regulations such as total duration and rest times.

The cost of the generated pairings is defined by several factors, which can include crew utilization efficiency, layover costs, and extended flight hours, which may result in overtime pay. However, when creating pairings, one must also take the robustness of the schedule into account. For example, if one were to minimize all the connecting times to maximize efficiency, it could result in not having large enough buffers for unforeseen events, which in turn could lead to delays propagating through the airline's schedule.

A simplified example of the input to the crew pairing problem can be seen in Table 2.1.

Index	Departure Airport	Time	Arrival Airport	Time	Deadhead
0	Gothenburg (GOT)	08:00	Stockholm (ARN)	9:15	No
1	Gothenburg (GOT)	09:00	Copenhagen (CPH)	10:00	No
2	Stockholm (ARN)	10:00	Copenhagen (CPH)	11:15	No
3	Copenhagen (CPH)	11:00	Gothenburg (GOT)	12:00	No
4	Hamburg (HAM)	14:15	Gothenburg (GOT)	15:30	No
5	Copenhagen (CPH)	12:15	Hamburg (HAM)	13:30	Yes

Table 2.1: Example Input for the Crew Pairing Problem. Flights marked with the deadhead column as "No", must be covered.

A straightforward solution to the input in Table 2.1, using Gothenburg as home base, would be to generate two pairings: $[0, 2, 5, 4]$ and $[1, 3]$, where the numbers correspond to the indices in the table. These two pairings are visualized in Figure 2.2.

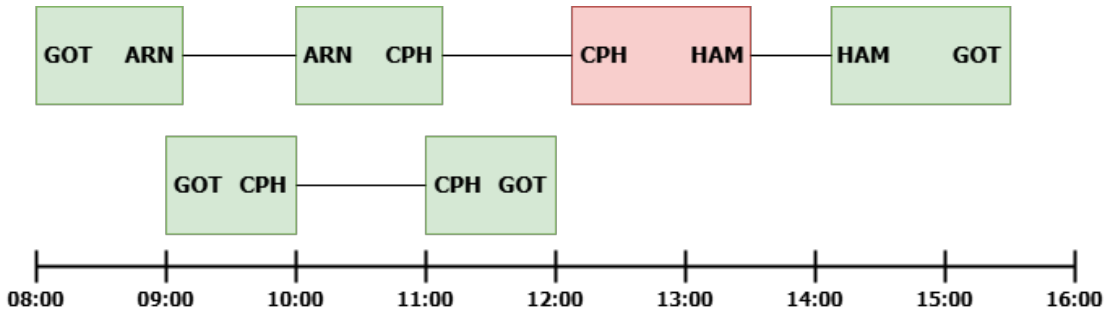


Figure 2.2: Example of pairings generated for the timetable in Table 2.1. The green color represents flight legs operated by crew, while the red color indicates a deadhead flight.

2.4 Modelling the Crew Pairing Problem as an Integer Linear Program

The crew pairing problem can be expressed as a set partitioning problem by defining:

- F: set of flight legs
- P: set of legal pairings
- c_p: cost associated with pairing p
- a_{ij}: equal to 1 if flight i is part of pairing j; equal to 0 otherwise
- x_j: decision variable, equal to 1 if pairing j is chosen; equal to 0 otherwise

.

The problem can then be modelled by the following ILP, where the goal is to choose the minimum-cost set of pairings (2.1) such that every flight is covered exactly once (2.2). To ensure the integrality of the problem, constraints (2.3) are used:

$$\min \sum_{j \in P} c_j x_j \tag{2.1}$$

$$\text{s.t.} \sum_{j \in P} a_{ij} x_j = 1, \qquad \forall i \in F \tag{2.2}$$

$$x_j \in \{0, 1\}, \qquad \forall j \in P \tag{2.3}$$

.

The defined ILP is a basic optimization model for the crew pairing problem. Often, airlines add additional crew balancing constraints to limit the total number of flying hours in the selected pairings [2]. It is also possible to model the problem as a set covering problem by changing the constraints in (2.2) to ≥ 1, which may result in overcovering, which could be solved by deadheading the extra crew.

2.4.1 Solution Approaches

Given the binary constraints on the decision variables, an integer solution approach such as branch-and-bound is needed [9]. However, the set of feasible pairings can be very large. With billions of possible pairings, solving the crew pairing problem using such approaches becomes practically infeasible.

Instead, approaches using column generation, see Section 2.6, are applied. However, the column generation algorithm is mainly used for Linear Programs (LPs). To enable the use of column generation for the crew pairing problem it is possible to:

1. Temporarily turn the ILP to a LP by further relaxing the constraints in (2.3) to $x_j \in \mathbb{R}$.
2. Solve the LP-relaxed problem using column generation.
3. Incorporate the branch-and-bound algorithm to handle the integrality constraints in the original ILP.

An algorithm that combines the ideas of LP-relaxations, column generation, and branch-and-bound to find an optimal integer solution is the branch-and-price algo-

rithm, which is commonly used in the crew pairing problem [12].

2.4.2 Boeing's Crew Pairing Optimizer

Today, Boeing's crew pairing optimizer uses branch-and-price in combination with connection fixing, which is described in [20]. This thesis is primarily focused on enhancing the column generation aspect of the crew pairing problem. As a result, the details of branch-and-price and connection fixing will not be addressed.

When solving large scale optimization problems, there is often a trade-off between achieving optimality and minimizing computational time. As the number of possible pairings increases to the billions, solving the problem to proven optimality can become computationally prohibitive. To address this, Boeing's optimizer can be adjusted to balance the trade-off between solution quality and computational efficiency. This can be achieved by introducing different approximations in the algorithm that enable the discovery of near-optimal solutions while significantly reducing computational time.

2.5 Dual Problem in Linear Programming

To solve LP-problems in general, and especially when using column generation, information from the dual problem is utilized. For any minimization problem, a lower bound on the objective value is desired, which can be obtained from the dual problem through the Weak Duality Theorem [4]. This theorem states that the objective value to any feasible solution of the dual problem is a bound on the objective value of any feasible solution to the original LP-problem, also called the primal. The dual problem can also be used to verify the optimality of a solution by using the Strong Duality Theorem. It states that when both the primal and dual have feasible solutions, the optimal values of their objective functions are equal.

The dual problem can be constructed by transforming the primal minimization problem into a corresponding maximization problem, where the constraints and objective function are reformulated. For example, given the primal problem

$$\min\ 5x_1+3x_2 \tag{2.4}$$
$$\text{s.t. } x_1 + x_2 \geq 1 \tag{2.5}$$
$$2x_1 + x_2 \geq 2 \tag{2.6}$$
$$x_1, x_2 \geq 0, \tag{2.7}$$

it is possible to obtain a lower bound by combining the inequalities in the constraints (2.5)-(2.6). By multiplying (2.6) with 2 and adding it to (2.5), a lower bound on the objective value is obtained:

$$5x_1 + 3x_2 \geq 5. \tag{2.8}$$

However, there may be another way to combine the constraints to get a better lower bound on the objective value.

To maximize the lower bound, the dual variables y_1 and y_2, corresponding to constraints (2.5) and (2.6), are introduced. Subsequently, the dual problem can be formulated by multiplying each dual variable with its corresponding constraint:

$$y_1(x_1 + x_2) \geq y_1 \tag{2.9}$$
$$y_2(2x_1 + x_2) \geq 2y_2. \tag{2.10}$$

By summing these constraints, a single inequality is obtained:

$$y_1(x_1 + x_2) + y_2(2x_1 + x_2) \geq y_1 + 2y_2, \tag{2.11}$$

which can be rewritten as:

$$x_1(y_1 + 2y_2) + x_2(y_1 + y_2) \geq y_1 + 2y_2. \tag{2.12}$$

The right-hand side of (2.12) creates a valid lower bound on the primal objective function (2.4), provided that the coefficients multiplied with x_1 and x_2 on the left-hand side of (2.12) are less than or equal to those in the primal objective function (2.4). For instance, if we set the dual variables to $y_1 = 1$ and $y_2 = 2$, we arrive once again at the inequality shown in (2.8). A worse lower bound is given by $y_1 = 1$ and $y_2 = 1$, which results in:

$$5x_1 + 3x_2 \geq 3x_1 + 2x_2 \geq 3, \tag{2.13}$$

since $x_1, x_2 \geq 0$. If $y_1 = 2$ and $y_2 = 2$ were chosen instead, one would obtain $6x_1 + 4x_2 \geq 6$, which would not be a valid lower bound as it is greater than the objective value. To maximize the lower bound, we formalize the idea into an LP and construct the dual problem:

$$\max\ y_1 + 2y_2 \tag{2.14}$$
$$\text{s.t. } y_1 + 2y_2 \leq 5 \tag{2.15}$$
$$y_1 + y_2 \leq 3 \tag{2.16}$$
$$y_1, y_2 \geq 0. \tag{2.17}$$

By solving the dual problem, which may be easier than solving the primal, we can obtain a bound on the objective value via the Weak Duality Theorem or verify optimality using the Strong Duality Theorem. To find an upper bound for a primal maximization problem, the same reasoning can be used to transform the primal into a corresponding dual minimization problem [4, 9].

2.5.1 Dual Problem for the Crew Pairing Problem

In general, the dual problem corresponding to the LP-relaxed set covering formulation of the crew pairing problem, presented in Section 2.4, is given by:

Primal Problem

$$\begin{aligned} \text{maximize} \quad & c^\top x \\ \text{subject to} \quad & Ax \geq b \\ & x \geq 0 \end{aligned}$$

Dual Problem

$$\begin{aligned} \text{minimize} \quad & b^\top y \\ \text{subject to} \quad & A^\top y \leq c \\ & y \geq 0, \end{aligned}$$

where $b = \mathbf{1}$.

2.6 Column Generation

Column generation is one of the most notable advances on the algorithmic front for solving large-scale models [11]. It is used in various optimization problems, particularly for large-scale linear programs, where the number of variables (columns) is extremely large.

Given the vast number of possible pairings in the crew pairing problem, each represented by a decision variable, the column generation approach is widely used. The optimal solution typically involves only a small subset of the possible pairings, making most of the pairings unnecessary to include in the optimization process.

In column generation, the original problem, called the Master Problem (MP), is solved by updating an implicit model iteratively, which reduces the computational complexity. The idea is to work with a smaller but sufficiently meaningful subset of variables, which forms the Restricted Master Problem (RMP) [10]. This process is illustrated in Figure 2.3.

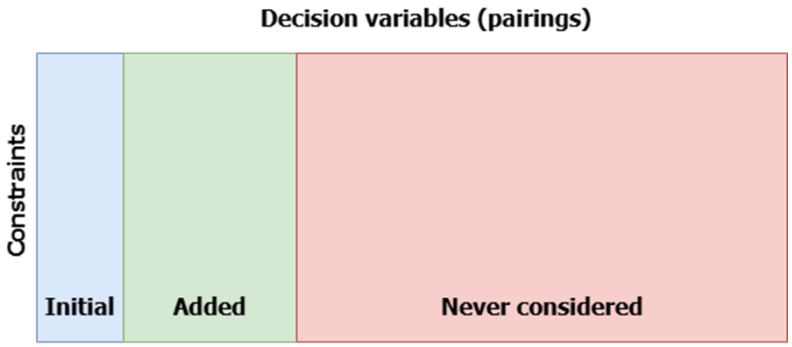


Figure 2.3: Illustration of how columns are generated and added to the RMP during CG, to reduce the computational complexity.

2.6.1 Restricted Master Problem

The Restricted Master Problem for the LP-relaxed set covering formulation of the crew pairing problem, presented in Section 2.4, is given by:

$$\min \sum_{j \in P'} c_j x_j \tag{2.18}$$

$$\text{s.t.} \sum_{j \in P'} a_{ij} x_j \geq 1, \qquad \forall i \in F \tag{2.19}$$

$$x_j \geq 0, \qquad \forall j \in P', \tag{2.20}$$

where P' is a subset of the possible pairings P.

The initial columns in the RMP are often chosen to create a feasible starting point for the iterative process. One common approach is to use a feasible solution heuristic, where a set of columns that forms a feasible solution to the MP is selected. The initial solution is often of low quality, and the initial columns are not typically used in the final solution.

2.6.2 Pricing Problem

To iteratively improve the solution, columns from the original set, P, must continually be added to the RMP. To decide which columns to bring into the RMP, the pricing problem is solved.

The pricing problem can be defined by first constructing the dual problem of the RMP:

$$\max \sum_{i \in F} y_i \tag{2.21}$$

$$\text{s.t.} \sum_{i \in F} a_{ij} y_i \leq c_j, \qquad \forall j \in P' \tag{2.22}$$

$$y_i \geq 0, \qquad \forall i \in F, \tag{2.23}$$

where y_i represents the corresponding dual variables for the constraints (2.19).

When deciding which pairings from the original set P to add to the RMP, we focus on the pairings that violate the constraints in (2.22). If a pairing has not been included in the RMP, their corresponding dual constraints are not a part of the dual problem. If a new pairing already respects the dual constraints (2.22), it implies that this pairing does not provide any new information or violation that would affect the dual value [17]. So, to be able to define a different combination of y_i, that increases the lower bound obtained from the dual problem, we should include pairings for which:

$$\sum_{i \in F} a_{ij} y_i - c_j > 0. \tag{2.24}$$

The inequality (2.24), can be rewritten into what is known as the reduced cost:

$$\hat{c}_j = c_j - \sum_{i \in F} a_{ij} y_i < 0. \tag{2.25}$$

The pricing problem can finally be defined as finding a new pairing $j \in P$, which is not already included in the RMP, such that $\hat{c}_j < 0$. Usually, the pairing with the largest negative reduced cost is added to the RMP, because it is the most promising in terms of improving the objective function.

2.6.2.1 Solving the Pricing Problem using a Shortest Path Algorithm

Given that the number of pairings in P can be huge, iterating through each pairing to find $\hat{c}_p$ is often infeasible. Instead, for each home base, the pairing with the most negative reduced cost can be found by solving a shortest path problem that exploit dominance characteristics in an pairing graph.

By subtracting the sum of dual variables corresponding to the constraints in (2.19) from the arcs' original weights, a multi-label shortest path algorithm can find the pairing with the most negative reduced cost. This is done by solving the multi-label shortest path problem where the sink and source node are the corresponding home base. This results in that many shortest path problems need to be solved, and each pairing found with a negative reduced cost will be added to the RMP. To reduce the number of subproblems that need to be solved, Boeing currently uses a k-shortest path algorithm to add the k pairings with most negative reduced cost during each iteration.

2.6.3 Column Generation Algorithm

During each iteration, promising pairings enter the RMP based on the reduced costs, calculated during the pricing problem. An iteration of column generation consists of:

(i) optimize the RMP to determine the current optimal value of the objective function, and the dual variables $\boldsymbol{y}$.

(ii) find, if there still is one, a variable with negative reduced cost during the pricing problem.

A summary of the algorithm can be seen in Figure 2.4.

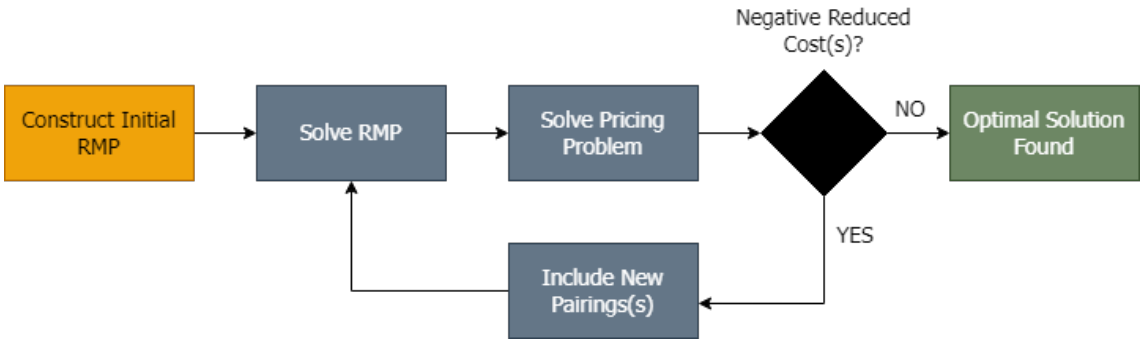


Figure 2.4: Summary of the Column Generation Algorithm.

2.7 Integrating Machine Learning into the Column Generation Algorithm

The integration of machine learning methods into column generation has shown some promise to increase the efficiency of solving large-scale optimization problems.

2.7.1 Previous Research

Most of the research focuses on the approach of using Reinforcement Learning (RL) to guide the selection of new columns during the pricing problem. A study by Morabit, Desaulniers, and Lodi presented a new approach, called column selection, to enhance the CG efficiency by formulating each iteration as a classification task [18]. Column selection applies a learned model to classify which of the generated columns during an iteration should be included, with the goal of reducing the computational complexity while re-optimizing the RMP. This resulted in up to 30% lower computing time for the vehicle and crew scheduling problem and the vehicle routing problem with time windows.

Another study by Cheng Chi et al. introduced RLCG, the first Reinforcement Learning approach for column generation [5]. This approach treats column generation as a sequential decision-making problem, by recognizing that the column selected in one iteration affects future selections. It is then possible to employ Graph Neural Networks to encode the state of the CG algorithm and the structure of the LP instance, to use Deep RL to select columns from the candidate columns from the pricing problem. By using the RLCG approach, the number of CG iterations for the Cutting Stock Problem and Vehicle Routing Problem with Time Windows, was reduced by 22.4% and 40.9%, respectively.

In addition to the work by Cheng Chi et al., several other studies have explored the use of RL in CG [19, 28, 16]. These studies further illustrate the growing interest of integrating machine learning methods in general, and RL methods in particular, into the column generation algorithm.

Limited research of replacing the pricing problem with a machine learning model has been done. Shen et al. proposed a Machine-Learning-based Pricing Heuristic (MLPH) designed to efficiently generate high-quality columns [22]. In each iteration of CG, MLPH utilizes a Support Vector Machine with linear kernel to predict the optimal solution of the pricing problem. Finally, a sampling method is used to generate multiple columns to include in the RMP. This approach showed to reduce the solving time for the Graph Coloring Problem, compared to other state-of-the-art methods.

2.7.2 Constructing the Initial RMP using Machine Learning

In contrast, the approach in this thesis is to use a neural network to generate pairings while constructing the initial RMP. As mentioned in Section 1.2, the pairings generated during the early rounds of column generation often does not contribute to the final solution. By integrating a neural network, the optimization process could be enhanced by immediately producing high-quality pairings, without the need for optimizer feedback through the dual variables.

After constructing the initial RMP using machine learning, it is possible to observe how the column generation process is affected. The study will both focus on how starting the algorithm with higher-quality pairings influences the number of generated pairings throughout the CG process and the quality of the final solution. The work can be regarded as a preliminary building block of integrating a machine learning model into the pricing problem for the crew pairing problem. If a model can predict high-quality pairings for the initial RMP, it should be possible to expand the model to also generate pairings during the CG iterations.

2.8 Machine Learning Techniques for Generating Pairings

To generate the pairings in the initial RMP, inspiration was taken from Natural Language Processing (NLP), and especially sequence generating tasks. This is done by conceptualizing each sentence as a pairing and each word within that sentence as a flight leg. The task of predicting high-quality pairings can thus be viewed similarly to sequence modeling in NLP, where the goal is to generate meaningful sentences from a given set of words.

2.8.1 Cosine Similarity

In NLP, it is often desired to calculate the similarity between sentences or words, and for that, distance measures are needed [15]. The cosine similarity is commonly used in high-dimensional spaces, such as feature matching or document classification, to indicate how similar two vectors are. If $\boldsymbol{x}$ and $\boldsymbol{y}$ are two feature vectors, then:

$$\text{cosine\_similarity}(\boldsymbol{x}, \boldsymbol{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|},$$

where $\cdot$ indicates the vector product, and $\|\mathbf{x}\|$ is the length of vector $\boldsymbol{x}$ [25]. As a result, the cosine similarity belongs in the interval $[-1, 1]$, where a 1 represents two proportional vectors, a 0 represents two orthogonal vectors, and a -1 represents two anti-parallel vectors.

In this thesis, the cosine similarity will be used to calculate the similarity between the feature representations of two different flight legs.

2.8.2 Special Tokens

In sequence generating tasks, tokens are often crucial for neural network models [1].

End-Of-Sequence (EOS) tokens are commonly used to signal to the model that a sequence, or as in this case, a pairing, should end [1]. By using EOS tokens, the model gets a clear indicator of when to stop producing flight legs for a pairing. The token is usually chosen as a reserved symbol that is distinct from the vocabulary. In this case, the EOS token will be chosen as a value that is distinct from the features created for each flight leg.

Additionally, to ensure that input sequences have the same length, padding tokens are commonly used [1]. Padding tokens enable the use of a masking layer in the model to signal to the sequence processing layers that certain time steps in the input are missing, and thus should be skipped during any calculations or processing of the data [24]. To ensure that not any relevant data is skipped, the padding token was chosen to lie outside the feature space of the flight data.

2.8.3 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are designed to process sequence-based data like text or flight legs in a pairing [21, 15]. RNNs have recurrent connections, allowing them to maintain a hidden state that can capture information from previous steps in the sequence. Given a sequence, $\boldsymbol{x} = (x_1, x_2, .., x_T)$, the hidden state, $\boldsymbol{h_t}$, is traditionally updated as:

$$\boldsymbol{h_t} = g(\boldsymbol{W}x_t + \boldsymbol{U}\boldsymbol{h_{t-1}}),$$

where g is an activation function, and $\boldsymbol{W}$ and $\boldsymbol{U}$ are weight matrices [7].

One of the main limitations in RNNs is the vanishing or exploding gradient problem. It has been observed that during the backpropagation process, gradients tend to either vanish or explode, which makes gradient based optimization methods struggle [3].

To reduce the negative impacts of this issue, several variants of RNNs have been developed, where Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRU) are the most prominent. Long Short-Term Memory networks were introduced by Hochreiter, Sepp & Schmidhuber, Jürgen in 1997 [14]. They are specifically designed to handle long-term dependencies by introducing the input, forget, and output gates which control the flow of information. They also introduce the cell state, which maintains a long-term memory that carries information across many steps. Gated Recurrent Units were introduced by Cho et al. in 2014 and offer a simplified alternative to LSTMs [6].

2.8.3.1 Gated Recurrent Units

Similar to LSTM networks, GRU networks have gating units that modulate the flow of information. They consist of reset and update gates, a hidden state, and a

candidate hidden state, see Figure 2.5 for an illustration [7].

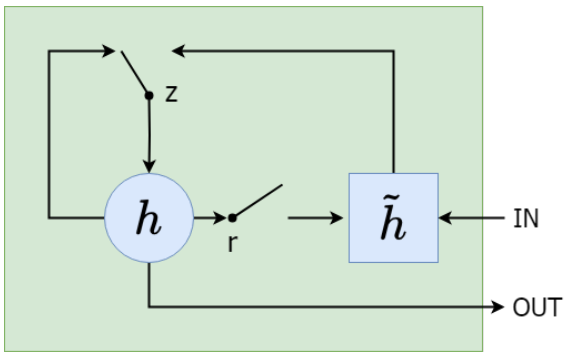


Figure 2.5: Illustration of the GRU, where r and z are the reset and update gates, and h and $\tilde{h}$ are the hidden and candidate hidden state.

The hidden state at time t, $\boldsymbol{h_t}$, is a linear interpolation between the previous hidden state and the current candidate hidden state, $\tilde{\boldsymbol{h}}_{\boldsymbol{t}}$:

$$\boldsymbol{h_t} = (1 - \boldsymbol{z_t}) \odot \boldsymbol{h_{t-1}} + \boldsymbol{z_t} \odot \tilde{\boldsymbol{h}}_{\boldsymbol{t}},$$

where $\odot$ represents the Hadamard product. The update gate, $\boldsymbol{z_t}$, decides how much the unit updates its hidden state, and is calculated as:

$$\boldsymbol{z}_t = \sigma(\boldsymbol{W}_z x_t + \boldsymbol{U}_z \boldsymbol{h}_{t-1}),$$

where σ is a logistic sigmoid function.

The candidate hidden state, $\tilde{\boldsymbol{h}}_{\boldsymbol{t}}$, is calculated similarly to the hidden state in an RNN:

$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W} x_t + \boldsymbol{U}(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1})).$$

The reset gates, $\boldsymbol{r_t}$, dictates how the candidate hidden state is affected by the previous hidden states. If $\boldsymbol{r_t}$ is set close to 0, it allows the unit to effectively forget the previously computed state, and makes it act as if it is reading the first symbol of an input sequence. The reset gates are calculated as:

$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}_r x_t + \boldsymbol{U}_r \boldsymbol{h}_{t-1}).$$

2.8.4 K-Means Clustering

K-Means clustering will be used to transform flight data into fixed-size, which is necessary for the data to be used as input to the model. It is one of the oldest and most used clustering algorithms that tries to find K non-overlapping clusters [26, 13]. Each cluster is represented by its centroid, which is typically the mean of the points in the cluster. The algorithm works by first selecting K initial centroids, which may be chosen at random or using some initialization method. Every data point is then assigned to the closest centroid, where each collection of points assigned to the same centroid forms a cluster. Finally, the centroids are recalculated and the

process repeats until no point changes cluster.

Given a data set:

$$X = [x_1,\ ..,\ x_n],$$

the K-means clustering algorithm can be expressed as:

$$\min_{\{m_k\},1\leq k\leq K} \sum_{k=1}^{K} \sum_{x\in C_k} \pi_x \operatorname{dist}(x, m_k), \tag{2.26}$$

where K is the user-defined number of clusters, π_x is the weight of x (typically 1), m_k is the centroid of cluster C_k, and the function "dist" computes the distance between a data point and a centroid. The function in (2.26) can be denoted as the loss function of K-means clustering. The centroids of the clusters, m_k, can be calculated as:

$$m_k = \frac{1}{n_k} \sum_{x\in C_k} \pi_x x,$$

where n_k is the number of data points assigned to cluster C_k. The choice of the distance function is optional but is usually chosen as the Euclidean distance.

K-means clustering is summarized in Algorithm 1.

Algorithm 1 K-means clustering

Initialize centroids $m_1, m_2, \ldots, m_K$
repeat
  for $i = 1, 2, \ldots, n$ **do**
    Assign x_i to cluster C_k, with $k = \arg\min_j \pi_{x_i} \operatorname{dist}(x_i, m_j)$
  end for
  for $k = 1, 2, \ldots, K$ **do**
    Update $m_k = \frac{1}{n_k} \sum_{x\in C_k} \pi_x x$
  end for
until Convergence of the loss in Equation (2.26) or maximum iterations reached

2.8.5 Principal Component Analysis

Principal Component Analysis (PCA) reduces the dimensionality of some high-dimensional data points by linearly projecting them onto a lower-dimensional space such that the reconstruction error made by the projection is minimal [27]. In this project, PCA will be utilized during the feature engineering phase of the flight data. In particular, it will be used to summarize matrices related to airport connections and flight times.

The dimensionality reduction can be achieved by using the mean-centered data set:

$$X = [\hat{x}_1,\ ..,\ \hat{x}_K],$$

where K is the number of data points, $\hat{x} = x - \mu$, and $\mu = \frac{1}{K}\sum_{k=1}^{K} x_k$. The covariance matrix of X can be calculated as:

$$C_{XX} = \frac{1}{K} X X^{\top}.$$

Subsequently, it is possible to find the directions of maximum variance by computing the eigenvalue decomposition of C_{XX}:

$$\frac{1}{K} X X^{\top} = C_{XX} = V D V^{\top},$$

where $D = \mathrm{diag}(\lambda_i)$, and each column in V is an eigenvector of C_{XX}. These eigenvectors are also known as the *principal components*. Large eigenvalues, λ_i, correspond to high variance. By choosing the submatrix with the eigenvectors V_m corresponding to the m largest eigenvalues, it is possible to project the data into the new, lower dimensional, subspace:

$$Y = V_m^{\top} X.$$

PCA is closely related to the Singular Value Decomposition (SVD), and is in fact nothing else than computing the SVD for the mean-centered data matrix [23, 13].

3
Methods

In this chapter, the methods used to address the defined problem are covered. First, the generation of the data set is explained. Then, the machine learning model used for generating the initial RMP is covered in depth.

3.1 Generating the Dataset

Boeing provided access to their crew pairing optimizer along with a set of artificial airline schedules or timetables. The schedules used in this research consist of a maximum of 333 flight legs, which are smaller than ones typically used in the crew pairing problem. The average number of flight legs in each schedule is 66 flight legs. The smaller schedules were chosen to simplify the model due to limited computational power. In total, 192 different schedules were used to train and test the model.

For each schedule, data corresponding to all flight legs in the timetable was extracted. Some rows from the resulting data frame for one of the schedules can be seen in Figure 3.1.

tab_ix	dep_airport	dep_time	dep_day	arr_time	arr_day	arr_airport	dh_code	occ
0	HAM	6:30	1	8:00	1	FRA	A	1
1	FRA	8:30	1	9:00	1	HAJ	A	1
2	HAJ	9:40	1	11:00	1	HAM	P	0

Figure 3.1: Example of extracted flight data from an airline's schedule.

By running the optimizer with the artificial timetables as input, it was possible to extract the generated pairings by the shortest path algorithm during the column generation process. After a final solution was found by the optimizer, the pairings that got accepted into the Master Problem was extracted. An extracted pairing consists of a list of indices, ordered sequentially, of the corresponding flight legs included in the pairing. For example, using the flight legs in Figure 3.1, the pairing [HAM → FRA, FRA → HAJ, HAJ → HAM] would be extracted as the list [0, 1, 2].

3.2 GRU-Enhanced Pairing Initializer

The machine learning model used to generate the initial RMP during this research is named the GRU-Enhanced Pairing Initializer (GEPI). GEPI's concept is to treat the extracted flight legs as a "vocabulary" for the corresponding schedule. Each schedule consists of a varying number of flight legs between different airports, resulting in a distinct vocabulary for each schedule. Therefore, the vocabulary was included as part of the model's input. Given that the sequential model used in this thesis requires a fixed-size input, the vocabulary is approximated through clustering. Clustering was chosen instead of padding the flight data to reduce GEPI's input size. Since many of the flight legs in the data are similar, clustering should not result in a significant loss of information.

The approximated vocabularies and the extracted pairings were used to train the model to replicate Boeing's crew pairing optimizer's shortest path algorithm. By using the generated pairings from GEPI as the initial columns in the RMP, the LP should be closer to the optimal basis and therefore lead to decreased computational time or lower cost solutions.

3.2.1 Feature Engineering

To effectively use the extracted flight data as input for a neural network, it is essential to represent the data numerically. This data preprocessing, known as feature engineering, is a critical step in representing the data for machine learning algorithms. For the simpler columns, such as deadhead code, occurrence count, and departure/arrival day, simple label encoding and scaling were done. These features are similar across the data in the different schedules, making them straightforward to encode. Label encoding was used to convert categorical values into numerical labels, while scaling ensured that numerical values were in a standard range.

3.2.1.1 Encoding Airports

Encoding the airports is more challenging, as both the airport names and the number of airports vary between different timetables. To handle this variability, a more complex approach is needed to ensure generalization across different timetables. To ensure that airports with similar characteristics in different schedules encode to similar values, the flight time matrix and the connectivity matrix were used.

By incorporating information about flight durations into the encoding process, airports connected by flights of similar duration should be mapped to similar values. Additionally, this approach should account for geographical information, as flights with similar flight times typically cover similar distances between airports. For airports with no direct connection between them, a large value in the flight time matrix was used. To summarize the flight time matrix, PCA was used to reduce the data to its two principal components. As a result, each airport is described by two features that reflect its flight time relative to other airports. An example of this process can be seen in Figure 3.2.

Flight Time Matrix

	HAM	FRA	HAJ	DUS	STR	MUC	LHR
HAM	0	90	80	999	999	999	180
FRA	90	0	30	60	999	999	999
HAJ	80	30	0	30	999	55	999
DUS	999	60	30	0	55	55	999
STR	999	999	999	55	0	30	999
MUC	999	999	55	55	30	0	999
LHR	180	999	999	999	999	999	0

Principal Components

	PCA1	PCA2
HAM	2.709935	-0.415461
FRA	0.848668	-1.719279
HAJ	-0.091679	-1.753917
DUS	-1.91339	-0.753064
STR	-2.028338	1.850288
MUC	-2.1504	0.542718
LHR	2.625204	2.248715

Figure 3.2: Flight Time Matrix (minutes) and its two Principal Components, resulting in two features for each airport.

The connectivity matrix represents the number of incoming and outgoing flights for the airports. This approach considers the network structure of the flights, and airports with a similar volume of flights should be encoded to similar values. The connectivity matrix is summarized using PCA in the same way as the flight time matrix, resulting in two features describing the number of flights between the airports. An example of this process can be seen in Figure 3.3.

Connectivity Matrix

	HAJ	HAM	FRA	LHR	DUS	MUC	STR
HAJ	0	12	0	0	20	0	0
HAM	0	0	12	12	0	0	0
FRA	19	0	0	0	0	0	0
LHR	0	12	0	0	0	0	0
DUS	0	0	12	0	0	6	0
MUC	8	0	0	0	0	0	8
STR	0	0	0	0	8	0	0

Principal Components

	PCA1	PCA2
HAJ	-2.151309	-1.853942
HAM	2.307373	-0.668061
FRA	-0.396017	1.855302
LHR	-1.04166	-0.709106
DUS	2.307373	-0.668061
MUC	-0.457862	2.283971
STR	-0.567898	-0.240104

Figure 3.3: Connectivity Matrix and its two Principal Components, resulting in two additional features for each airport.

Additionally, a ranking for each airport was calculated based on the total number of outgoing and incoming flights, which was used to sort the approximated vocabulary. An example of a final encoding for the airport Hamburg (HAM) can be found in Figure 3.4.

	Time_PCA1	Time_PCA2	Connectivity_PCA1	Connectivity_PCA2	Ranking
HAM	2.709935	-0.415461	2.307373	-0.668061	2

Figure 3.4: Final airport encoding for Hamburg (HAM).

Together, these five features provide a representation of each airport's connectivity and flight time characteristics and should enable the neural network used in GEPI to learn the underlying relationships between airports in different schedules.

3.2.1.2 Encoding Time

As all flight legs in the dataset are scheduled to depart and arrive at exact minutes, the arrival and departure times were converted into minutes after midnight. Keeping the minutes after midnight as a feature creates some challenges due to the cyclical nature of time. For instance, a flight departing a minute before midnight would be encoded very differently from one departing a minute after midnight.

Instead, the time was encoded using a cyclical approach with sine and cosine transformations. By taking advantage of these transformations' cyclic properties, it is possible to create two features that more accurately reflect the periodic nature of time. The process begins by calculating the time as a fraction of the day, subsequently, the features are computed as follows:

$$\begin{aligned} \text{minutes\_cos} &= \cos(2\pi \cdot \text{time\_fraction}) \\ \text{minutes\_sin} &= \sin(2\pi \cdot \text{time\_fraction}). \end{aligned}$$

The results of these transformations for various times are illustrated in Figure 3.2, which represents the unit circle. For instance, the encoding for noon (12:00) can be determined by first calculating the fraction of the day as 0.5. Multiplying 0.5 with 2π corresponds to the angle π on the unit circle, resulting in the coordinates and the final encoding (-1, 0).

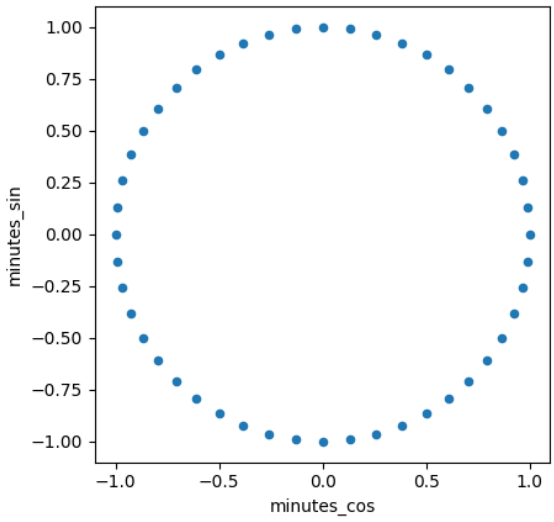


Figure 3.5: Encoding of time using sine and cosine transformations on the unit circle.

3.2.2 Clustering of the Flight Data to Approximate the Vocabulary

As outlined in Section 3.2, the encodings of the extracted flight legs function as a vocabulary for the corresponding schedule. To utilize this vocabulary as input

for GEPI, it must be transformed into a fixed-size format. This transformation is accomplished using K-means clustering, chosen due to its speed, efficiency, and interpretability. By applying K-means to the preprocessed flight data, flight legs with similar feature vectors should be grouped together, which reduces the overall data dimensions to a fixed input size, see Figure 3.6.

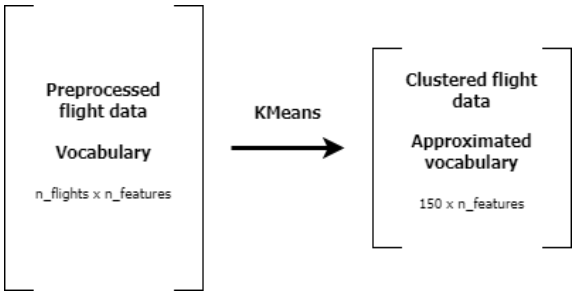


Figure 3.6: Clustering of the preprocessed flight legs to a fixed input size. Each row in the vocabulary contains the feature vector for a flight leg.

3.2.2.1 Permutation Invariance and Neural Networks

Permutation invariance refers to the property of a function where the output remains unchanged when the input elements are rearranged. A function f is said to be permutation invariant if:

$$f(x_1, x_2, \ldots, x_n) = f(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$$

for any permutation π. This property is particularly desirable for the input vocabulary to the model, as it ensures that the order of the cluster centers in the approximated vocabulary does not affect the predicted flight.

However, GEPI processes the input data sequentially and is therefore not inherently permutation invariant. To address this limitation and ensure that the output of the model is not dependent on the order of the extracted flights, the cluster centers are ordered based on the airport rankings, calculated in section 3.2.1.1. By establishing a consistent ordering for the input, we achieve some standardization and the model should process similar timetables in a consistent manner, aiding generalization between schedules.

3.2.3 Model Description

The input of GEPI consists of two concatenated matrices. The first matrix is the approximated vocabulary, representing the available flights legs in the schedule. The second matrix represents the current incomplete pairing, consisting of the features of the corresponding flight legs. Each row in this matrix contains the features of an individual flight leg. The maximum length of a pairing is 15 flight legs, chosen as this exceeds the length of any extracted pairing generated during the optimization process for the artificial schedules. If the current pairing contains less than 15 flight legs, the remaining rows are filled with a padding token. The model is used to

predict the feature vector of the next flight leg in the pairing, resulting in an output of size $1 \times$ n_features. This process is visualized in Figure 3.7.

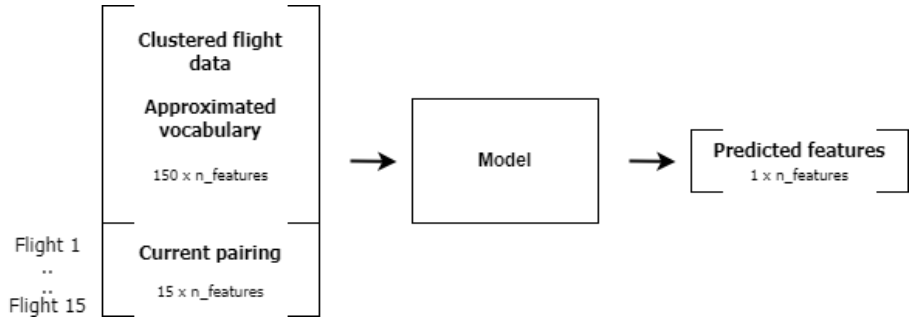


Figure 3.7: Prediction of the feature vector of the next flight leg in the pairing using two concatenated matrices as input. The first matrix represents the approximated vocabulary, and the second matrix represents an incomplete pairing.

The input pairing matrix is initialized with the features of one flight leg, with the remanding 14 rows set to a padding token. The model is then iteratively used to:

1. Predict the feature vector of the next flight leg.
2. Pick the flight leg with the most similar feature vector from the full-size vocabulary using cosine similarity.
3. Update the input pairing matrix by inserting the feature vector of the most similar flight leg into the corresponding row.

This process is repeated until an EOS token is predicted or the pairing reaches its maximum length of 15 flights. An example for the first iteration is shown in Figure 3.8.

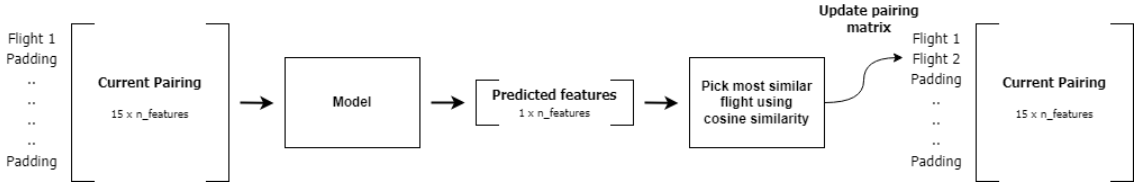
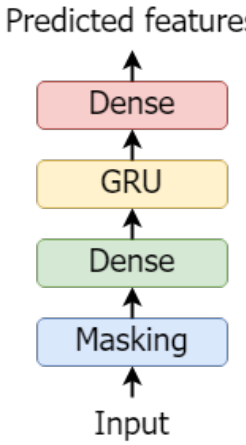


Figure 3.8: Visualization of the first iteration of predicting a full pairing using GEPI. For clarity, the approximated vocabulary has been removed in this visualization.

By changing the initial flight in the input pairing, it is possible to use GEPI to predict several different pairings that can be used as the initial columns in the RMP.

3.2.4 Model Architecture

GEPI employs a sequential neural network model with a GRU as its core component. A detailed view of the model can be seen in Figure 3.9.



(a) Layers configuration

	Layer (type)	Output Shape	Param #
0	Input	(None, 165, 19)	0
1	Masking	(None, 165, 19)	0
2	Dense	(None, 165, 19)	380
3	GRU	(None, 250)	203250
4	Dense	(None, 19)	4769
5	Total params:		208,399 (814.06 KB)
6	Trainable params:		208,399 (814.06 KB)

(b) Model layers and parameters summary

Figure 3.9: Detailed view of the sequential model used in GEPI, including its layers and parameter summary.

The model incorporates a masking layer to mask any padding tokens present in the input pairing matrix, as described in Section 3.2.3. This enables the model to handle variable-length pairings during training by disregarding the padding tokens in any computations, thereby allowing the model to focus on the relevant data.

Before passing the masked input to the GRU, it is processed by a fully connected dense layer. This layer applies an initial learned linear transformation followed by a non-linear activation function, enhancing the representation of the data. The dense layer introduces non-linearity in the network, which can help to learn complex patterns and relationships in the approximated vocabulary, further aiding generalization between schedules. The number of trainable parameters in the dense layer is calculated using the formula:

$$\begin{aligned} \text{Params} &= \underbrace{(\text{Input Units} \times \text{Output Units})}_{\text{weights}} + \underbrace{\text{Output Units}}_{\text{biases}} \\ &= (19 \times 19) + 19 = 380 \text{ params.} \end{aligned}$$

The transformed input data is then processed by the main component, the GRU, which has 250 units, a dropout rate of 0.1, and uses the hyperbolic activation function. This results in the total number of trainable parameters for the update gate, the reset gate, and the candidate hidden state:

$$\text{Params} = 3 \times \Bigg[\underbrace{(\text{Input Units} \times \text{Output Units})}_{\text{Input weights}} + \underbrace{(\text{Output Units} \times \text{Output Units})}_{\text{Recurrent weights}} + \underbrace{2 \times \text{Output Units}}_{\text{biases}} \Bigg]$$
$$= 3 \times [(19 \times 250) + (250 \times 250) + 2 \times 250] = 203250 \text{ params}$$

Finally, the output from the GRU is passed to a second dense layer which transforms it into a vector of shape $1 \times$ n_features, using a linear activation function. This vector represents the features of the predicted next flight leg in the pairing. The number of trainable parameters for this layer is calculated similarly to the previous dense layer, resulting in $(250 \times 19) + 19 = 4769$ parameters.

3.2.5 Training the model

The model was trained using the dataset described in Section 3.1, with the extracted pairings as targets. The model was fitted using the Adam optimizer and the cosine similarity loss with a batch size of 64. In total, 80 out of the 192 schedules, chosen randomly, were used during the training process.

Each extracted pairing was converted into a sequence of input-target pairs. For example, given a pairing with flight leg indices [0, 3, 7, 2], the training data was structured as shown in Table 3.1. Importantly, not the actual flight indices were used in the input pairing matrix and as targets, but rather the features of the corresponding flight leg.

Input Pairing Matrix	**Target**
0	3
0, 3	7
0, 3, 7	2
0, 3, 7, 2	EOS

Table 3.1: Example of input-target pairs generated from the extracted data.

The EOS is a vector of size $1 \times$ n_features, that signals that the pairing should end and that no more flight should be predicted. The EOS vector consists of -1s, chosen to be distinct from any feature vector in the vocabulary.

3.2.6 Integration of GEPI into Boeing's Optimizer

GEPI was integrated into Boeing's crew pairing optimizer by adding three additional steps. First, the optimizer initializes as usual by reading and processing the required input data, such as the timetable and rulesets. After the initial setup, the process moves on to feature engineering, where the flight data is transformed. This

is followed by K-means clustering, which is used to create an approximated vocabulary of fixed size. Subsequently, the approximated vocabulary is used to predict the pairings for the initial RMP, aiming to generate high-quality pairings immediately. Finally, the optimizer proceeds to the column generation process as usual. These steps are summarized in Figure 3.10.

Figure 3.10: Summary of the integration of GEPI into Boeing's optimizer. The steps highlighted in green represent the new procedures introduced by GEPI, while the blue boxes indicate the original steps in Boeing's optimizer's workflow.

3.3 Optimizer Configuration and Parameter Settings

As presented in Section 2.4.2, Boeing's optimizer can be adjusted to balance the trade-off between solution quality and computational efficiency. The schedules used during this research are small and could potentially be solved to proven optimality by the optimizer, limiting GEPI's potential solely to enhancing computational time. Therefore, during the tests, the optimizer was adjusted to emphasize computational time, which may allow GEPI to improve both solution quality and computational time.

4
Results

In this chapter, the results of the project are presented and compared to Boeing's crew pairing optimizer. GEPI was used to generate the initial RMP after which Boeing's optimizer was employed, as described in Section 3.2.6. As outlined in Section 3.3, the optimizer was adjusted to prioritize computational time. The solutions produced from the optimizer are static, which allows for comparisons between the solutions produced with and without using GEPI to generate the initial RMP. All 192 available schedules were used to test GEPI. The schedules in the training data were included in the testing phase as the main interest in this research is not the predictive accuracy of GEPI itself, but rather its impact on the CG process and on the final solution.

In Section 4.1, the accuracy of GEPI is presented. Subsequently, the impact on the CG process and on the final solutions of using GEPI are presented in Section 4.2-4.3. Finally, the computational performance is compared in Section 4.4.

4.1 Accuracy of GEPI on Testing Data

The average accuracy and the average cosine similarity were calculated for each schedule excluded from the training data, see Table 4.1. The accuracy indicates how often the model correctly predicts the next flight leg in an extracted paring.

Avg. accuracy	39.94 (18.13)
Avg. cosine similarity	0.79 (0.19)

Table 4.1: Average accuracy and cosine similarity of GEPI for each schedule on the testing data. The value in the parentheses refers to the standard deviation.

4.2 Impact of GEPI with Illegal and Incomplete Pairings

GEPI was used to generate the pairings in the initial RMP. By allowing the optimizer to add any pairing, including illegal or incomplete pairings, generated by the model, 78/192 schedules ended up with a lower final cost. In this case, an illegal pairing usually consists of a sequence of flight legs with not enough rest time in

between flights.

The average total cost for the 78 schedules with an improved solution decreased by 23.5%, with a standard deviation of 10.12 and a median of 17%. Out of these schedules, 47 were included in the training data, while 31 schedules were unseen. An example of the final solution, with and without using GEPI to generate the initial columns, for one of the schedules can be seen in Figure 4.1.

	Description	Value
0	Generated trips	1172
1	Complete illegal trips	0
2	TOTAL cost	1540
3	Cost of complete trips	1540
4	Working days	15
5	Average block time per working day	2:48
6	Deadhead time	4:25
7	Duty time	105:45
8	Duty time per working day	7:03
9	Duty time/block time	2.508
10	Duties	15
11	Average duty time per duty	7:03

(a) Without GEPI

	Description	Value
0	Generated trips	1072
1	Complete illegal trips	3
2	TOTAL cost	1120
3	Cost of complete trips	1120
4	Working days	11
5	Average block time per working day	3:50
6	Deadhead time	1:50
7	Duty time	56:55
8	Duty time per working day	5:10
9	Duty time/block time	1.350
10	Duties	11
11	Average duty time per duty	6:40

(b) With GEPI

Figure 4.1: Summary of the solutions produced for one schedule while allowing illegal and incomplete pairings in the initial RMP. In subfigure (a) Boeing's optimizer was used. In subfigure (b) GEPI was used to produce the initial pairings, which may be illegal or incomplete, before employing Boeing's optimizer.

In the example schedule in Figure 4.1, the number of generated trips during column generation decreased from 1172 to 1072. The total cost of the solution decreased from 1540 to 1120. However, when using GEPI, the solution includes 3 complete but illegal trips, which likely account for the reduction in cost.

4.3 Impact of GEPI with Legal and Complete Pairings

In this section, the impact of using GEPI, while only allowing legal and complete pairings to be included in the initial RMP, will be presented.

4.3.1 Lower-Cost Solutions with GEPI-Generated Pairings

If illegal and incomplete pairings are filtered out before adding them to the initial RMP, 34/192 schedules resulted in a lower final cost. The average final cost was 15.6% lower, with a standard deviation of 4.18 and a median of 17%. Out of these schedules, 22 schedules were included in the training data, while 12 were unseen. An

example of the final solution, with and without using GEPI to generate the initial columns, for one of the schedules can be seen in Figure 4.2.

	Description	Value
0	Generated trips	572
1	Complete illegal trips	0
2	TOTAL cost	880
3	Cost of complete trips	880
4	Working days	7
5	Average block time per working day	3:00
6	Deadhead time	0:00
7	Duty time	48:10
8	Duty time per working day	6:52
9	Duty time/block time	2.285
10	Duties	5
11	Average duty time per duty	9:38

(a) Without GEPI

	Description	Value
0	Generated trips	378
1	Complete illegal trips	0
2	TOTAL cost	780
3	Cost of complete trips	780
4	Working days	6
5	Average block time per working day	3:30
6	Deadhead time	0:00
7	Duty time	40:20
8	Duty time per working day	6:43
9	Duty time/block time	1.913
10	Duties	7
11	Average duty time per duty	5:45

(b) With GEPI

Figure 4.2: Summary of one of the solutions produced for one schedule while filtering out illegal and incomplete pairings from the initial RMP. In subfigure (a) Boeing's optimizer was used. In subfigure (b) GEPI was used to produce the initial pairings before employing Boeing's optimizer.

4.3.2 Higher-Cost Solutions with GEPI-Generated Pairings

In two cases out of the 192 schedules, using GEPI to create the initial RMP led to a worse solution. A summary for one of those schedules can be seen in Figure 4.3.

	Description	Value
0	Generated trips	602
1	Complete illegal trips	0
2	TOTAL cost	218
3	Cost of complete trips	218
4	Working days	8
5	Average block time per working day	2:38
6	Deadhead time	1:30
7	Duty time	53:10
8	Duty time per working day	6:38
9	Duty time/block time	2.522
10	Duties	7
11	Average duty time per duty	7:35

(a) Without GEPI

	Description	Value
0	Generated trips	328
1	Complete illegal trips	0
2	TOTAL cost	280
3	Cost of complete trips	280
4	Working days	10
5	Average block time per working day	2:06
6	Deadhead time	3:00
7	Duty time	53:55
8	Duty time per working day	5:23
9	Duty time/block time	2.557
10	Duties	10
11	Average duty time per duty	5:23

(b) With GEPI

Figure 4.3: Summary of a schedule that led to a worse solution by using GEPI to construct the initial RMP. In subfigure (a) Boeing's optimizer was used. In subfigure (b) GEPI was used to produce the initial pairings before employing Boeing's optimizer.

4.3.3 GEPI-Generated Pairings Count

The number of legal and complete pairings that were generated by GEPI to construct the initial RMP for the schedules is summarized in Table 4.2.

Metric	Number of legal and complete pairings
Average	7.37 (9.97)
Maximum	86
Minimum	0

Table 4.2: Summary of the number of legal and complete pairings generated by GEPI across 192 different schedules. The table shows the average number of pairings per schedule, the highest number of pairings generated for a single schedule, and the lowest number of pairings generated for any schedule. The value in parentheses refers to the standard deviation.

One can evaluate the performance of GEPI in generating complete pairings by analyzing the proportion of pairings that are legal but not necessarily complete. The number of legal pairings that were generated while constructing the initial RMP for the schedules is summarized in Table 4.3. Note that some of the pairings in the table might be incomplete and filtered out from the RMP before the column generation process is started.

Metric	Number of legal pairings
Average	15.84 (23.04)
Maximum	224
Minimum	0

Table 4.3: Summary of the number of legal pairings generated by GEPI across 192 different schedules. Note that some of these pairings might be incomplete and filtered out from the RMP before the column generation process is started. The table shows the average number of pairings per schedule, the highest number of pairings generated for a single schedule, and the lowest number of pairings generated for any schedule. The value in parentheses refers to the standard deviation.

4.3.4 Retention of GEPI-Pairings in the Final Solution

A total of 59 schedules included pairings generated by GEPI in the final solution. See Table 4.4 for a summary of the distribution of schedules by the number of pairings.

Number of GEPI-Pairings in Final Solution	Number of Schedules
0	133
1	51
2	7
7	1

Table 4.4: Distribution of schedules by number of GEPI-generated pairings used in the final solution.

4.3.5 GEPI's Impact on the Column Generation Process

While using GEPI to generate the initial RMP, the number of pairings generated by Boeing's optimizer during the pricing problems throughout the column generation process changed for 184 schedules. A summary of the changes can be seen in Table 4.5.

Metric	Number of pairings
Average change	1.94 (94.16)
Maximum increase	581
Maximum decrease	-431

Table 4.5: Summary of how the number of generated pairings during the pricing problems throughout the CG process is impacted by using GEPI to construct the initial RMP. The table shows the average number of pairings per schedule, the largest increase, and the largest decrease in the number of pairings. The value in the parentheses refers to the standard deviation.

Out of the 184 schedules, 38 had a decreased number of generated pairings by the shortest path algorithm. The average decrease was 104.82 pairings with a standard deviation of 109.82. For the 146 schedules with an increased number of generated pairings, the average increase was 29.73 pairings with a standard deviation of 65.55.

A summary of how the number of generated pairings by Boeing's optimizer throughout the column generation process for the 34 schedules with lower final cost changed can be seen in Table 4.6.

Metric	Number of pairings
Average change	-35.11 (189.68)
Maximum increase	581
Maximum decrease	-407

Table 4.6: Summary of how the number of generated pairings during the pricing problems throughout the column generation process is impacted by using GEPI to construct the initial RMP. The table shows the average number of pairings per schedule, the largest increase, and the largest decrease in the number of pairings. Only the schedules with a lower final cost included. The value in the parentheses refers to the standard deviation.

4.4 Computational Performance

The computational performance of running the optimizer with and without GEPI was measured. In 37 schedules, using GEPI led to slower convergence times, with execution time increasing between 20% and 4000%. In contrast, 3 schedules exhibited faster performance, achieving around a 50% reduction in execution time compared to not using GEPI. A total of 176 schedules displayed increased memory consumption, with 148 of these experiencing a rise between 10% and 15%. Additionally, 16

schedules demonstrated decreased memory usage, with reductions ranging from 10% to 20%.

5
Discussion

In this chapter, the results are examined, key findings highlighted, and future work outlined. First, the results are discussed and analyzed in Sections 5.1-5.4. Subsequently, the key findings are summarized in Section 5.5. Finally, directions for future work are outlined in Section 5.6.

5.1 Impact of Dataset on Results and Accuracy

The extracted pairings can contain overlapping sequences, given that two pairings can have the same subsequence of flight legs. When the same input is associated with different outputs, the model receives conflicting signals during training. For example, two pairings can have the same initial flight sequences: [0, 3, 1] and [0, 3, 6, 2], where the numbers refer to indices of flight legs. If the model is used to predict the flight leg after [0, 3], it might predict the features of the flight leg with index 1. This would result in a correct prediction for the first pairing, but an incorrect prediction for the second pairing, even though the next flight leg is legal. This reduces the prediction accuracy and the cosine similarity on the testing data and could explain the results in Section 4.1. However, having multiple outputs for the same input could also have a regularization effect by preventing the model from overfitting to a single output. This could make the model predict a more general representation of the flight legs, that minimizes the cosine similarity loss across all instances of the same input. This averaged prediction might not be the same as in the extracted pairing but might represent another legal flight leg.

The accuracy could also be impacted by the fact that the same flight leg can exist in the data, but with different departure times. For instance, a flight may have departure times available at 5-minute intervals between 11:00 and 11:30 and the algorithm is required to select one of these departure times. If GEPI predicts a flight leg at 11:20, while the extracted pairing included the flight at 11:25, it would be classified as an incorrect prediction, even though it might create a legal and high-quality pairing.

5.1.1 Artificial Set of Schedules

The schedules used in this research are smaller than the ones typically used in the crew pairing problem, as mentioned in Section 3.1. The solutions of the smaller schedules typically have a lower total cost, which impacts the percentage change by

using the model. The results when using the model to generate the initial RMP showed a median cost decrease of 17%, which represents a cost change from 12 to 10. However, for larger schedules, one could expect a lower percentage-wise change in the total cost. For larger problems, the column generation process involves many more iterations over a much larger solution space, which could decrease the impact of the initial RMP. To explore the impact of using the model on larger schedules, changes to the model must be made, which is covered in Section 5.6.

5.2 Computational Performance

No optimizations of the code were done regarding the computational performance, as the research primarily focused on the impact of GEPI on the CG process and on the final solution. Almost all schedules showed an increase in memory consumption, some of which could be explained by extensive use of logging files. However, some of the schedules showed a decrease in memory consumption, indicating some potential for the model to decrease resource usage during optimization.

5.3 Generating Legal and Complete Pairings

The results in Section 4.2 indicate that GEPI is able to predict attractive pairings to the optimizer, decreasing the total cost for 41% of the schedules. However, the generated pairings might be illegal or incomplete and not allowed in an actual solution, which explains some of the reduction in cost. In Section 4.3.1, while filtering out invalid pairings during the construction of the initial RMP, there was a decrease in the total cost for 18% of the schedules. This suggests that GEPI is able to generate an initial set of legal and complete pairings that improves the solution quality for some schedules.

The number of generated legal pairings in Table 4.3 is approximately double the number of legal and complete pairings in Table 4.2. This shows that the model has difficulty deciding when a pairing should end, resulting in the generation of legal but incomplete pairings. One reason could be that the chosen EOS token is not distinct enough from the possible flight legs in the vocabularies. Other possible reasons could be that the EOS token is rarer than real flight legs in the training data, or that the loss function might not adequately penalize incomplete sequences.

Table 4.4 shows that 59 schedules used GEPI-generated legal and complete pairings in its final solution. However, this did not lead to a reduction in the final cost for all of these schedules. This suggests that even if the initial RMP is of higher quality, it is not guaranteed to improve the final solution.

5.4 Optimizer Configuration and Parameter Settings

As mentioned in Section 3.3, the optimizer was adjusted to prioritize computational time to allow GEPI to improve both solution quality and computational time. The results in Section 4.4 demonstrates that GEPI is currently not able to reduce the computational time of the optimization process, as discussed in Section 5.2. The results presented in Section 4.3 indicate that GEPI can enable the optimizer to produce higher quality solutions. However, as the optimizer is adjusted to prioritize optimality, GEPI's impact on solution quality is expected to diminish. To understand GEPI's impact on computational time and solution quality under different optimizer settings, further tests and analysis are needed.

5.5 Conclusion

By using a combination of feature engineering and clustering techniques it is possible to extract and preprocess data from Boeing's optimizer to leverage a GRU model. The results indicate that GEPI is able to predict attractive pairings while constructing the initial RMP. A lower total cost was achieved for 34 schedules, where 12 of these were not included in the training data, indicating the model is able to generalize between data from different airlines.

The results in Table 4.5 show that on average, almost no change is made in the number of pairings generated during the rest of the column generation process. If only the schedules with a lower total cost from using the model are included, the number of pairings generated during the CG process decreased on average, see Table 4.6. This suggests that if the generated initial RMP is of high quality, the total cost tends to decreases and the CG process shortens. However, it is important to note the high variance in both tables, which indicates inconsistency in how the CG process is affected by using the model.

In conclusion, the results demonstrate the potential of using a machine learning approach to enhance the optimization process. The trained model can be regarded as a preliminary building block for integrating a machine learning model throughout the iterations of the CG algorithm for the crew pairing problem. To gain a more comprehensive understanding of GEPI's potential and limitations, further tests and analysis are needed.

5.6 Future Work

Little work was done to fine-tune the model due to limited computational power. An immediate next step would be to experiment with different activation functions, the number of units in the GRU layer, and different clustering dimensions while approximating the vocabulary. To handle larger schedules effectively, a larger number

of clusters would likely be necessary to preserve more of the flight data.

To address the issue of predicting illegal or incomplete trips, one could incorporate penalty terms in the loss function. This would be possible by using Boeing's legality checks during training to punish predicted pairings that do not adhere to predefined constraints. Another approach would be to make use of constrained structured prediction. By integrating constraints directly into the prediction process, it would be possible to ensure that the generated pairings adhere to specific rules or conditions.

To further advance this foundational work and integrate neural networks into the pricing problem during CG iterations, several modifications to the model will be necessary. To predict pairings during the pricing, more information from the RMP would need to be included in the input of the model. A first step could be to encode the current state of the RMP, including already added pairings and the dual values.

A final step would be to try out different models and evaluate their performance and identify the most effective approach for predicting pairings during the pricing problem. This could include experimenting with other RNNs, such as LSTM networks or transformer-based architectures.

Bibliography

[1] Atçılı, A., Özkaraca, O., Sarıman, G., & Patrut, B. (2023). Next Word Prediction with Deep Learning Models. In Smart Applications with Advanced Machine Learning and Human-Centred Problem Design. ICAIAME 2021. Engineering Cyber-Physical Systems and Critical Infrastructures, vol 1. Springer, Cham. https://doi.org/10.1007/978-3-031-09753-9_38

[2] Belobaba, P., Odoni, A., & Barnhart, C. (2009). *The Global Airline Industry*. Wiley.

[3] Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks, 5(2), 157-166. https://doi.org/10.1109/72.279181

[4] Birge, J. R., & Louveaux, F. (2011). *Introduction to Stochastic Programming*. Springer Series in Operations Research and Financial Engineering. Springer, New York, NY. https://doi.org/10.1007/978-1-4614-0237-4

[5] Chi, C., Aboussalah, A. M., Khalil, E. B., Wang, J., & Sherkat-Masoumi, Z. (2023). A Deep Reinforcement Learning Framework For Column Generation. arXiv. https://doi.org/10.48550/arXiv.2206.02568

[6] Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. arXiv preprint arXiv:1406.1078. https://doi.org/10.48550/arXiv.1406.1078

[7] Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. The Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI-22). https://doi.org/10.48550/arXiv.1412.3555

[8] Cook, A., & Goodwin, P. (2010). *Airline Operations and Scheduling, Second Edition*.

[9] Desaulniers, G., Desrosiers, J., & Solomon, M. M. (2005). *Column Generation*. In *A Primer in Column Generation*

[10] Desaulniers, G., Desrosiers, J., & Solomon, M. M. (2005). *Column Generation*. In *A Primer in Column Generation*, Chapter 1.

[11] Desaulniers, G., Desrosiers, J., & Solomon, M. M. (2005). *Column Generation*. In *Large-Scale Models In The Airline Industry*, Chapter 6.

[12] Dück, V., Wesselmann, F. & Suhl, L. Implementing a branch and price and cut method for the airline crew pairing optimization problem. Public Transp 3, 43–64 (2011). https://doi.org/10.1007/s12469-011-0038-9

[13] Heinlein, A. (2023). Unpublished lecture notes for Linear Algebra and Optimization for Machine Learning (WI4635). Electrical Engineering, Mathematics and Computer Science, Delft University of Technology.
[14] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735
[15] Kulkarni, A., Shivananda, A., & Kulkarni, A. (2022). Natural Language Processing Projects. Apress. https://doi.org/10.1007/978-1-4842-7386-9
[16] Lindén, A. (2023). Guiding Column Generation using Deep Reinforcement Learning. Master's thesis, Chalmers University of Technology.
[17] Lopes Dos Santos, B. F. (2023). Unpublished lecture notes for Airline Planning and Optimisation (AE4423-20). Aerospace Engineering, Delft University of Technology.
[18] Morabit, M., Desaulniers, G., & Lodi, A. (2021). Machine-Learning–Based Column Selection for Column Generation. Transportation Science, 55(4), 815-831. https://doi.org/10.1287/trsc.2021.1045
[19] Parsonson, C. W. F., Laterre, A., & Barrett, T. D. (2022). Reinforcement Learning for Branch-and-Bound Optimisation using Retrospective Trajectories. arXiv. https://doi.org/10.48550/arXiv.2205.14345
[20] Potter, D. (2008). An Implementation of a Constraint Branching Algorithm for Optimally Solving Airline Crew Pairing Problems. Master's thesis, Chalmers University of Technology.
[21] Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning representations by back-propagating errors. Nature, 323, 533–536. https://doi.org/10.1038/323533a0
[22] Shen, Y., Sun, Y., Li, X., Eberhard, A., & Ernst, A. (2022). Enhancing Column Generation by a Machine-Learning-Based Pricing Heuristic for Graph Coloring. The Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI-22). https://doi.org/10.48550/arXiv.2112.04906
[23] Shlens, J. (2005). A Tutorial on Principal Component Analysis. Systems Neurobiology Laboratory, Salk Institute for Biological Studies and Institute for Nonlinear Science, University of California, San Diego. Retrieved from https://www.cs.cmu.edu/ elaw/papers/pca.pdf
[24] TensorFlow. (n.d.). Understanding masking and padding. Retrieved from https://www.tensorflow.org/guide/keras/understanding_masking_and_padding
[25] Tan, P.-N., & Steinbach, M. (2019). Introduction to Data Mining, Global Edition. Pearson Education Limited. ISBN: 9780273769224.
[26] Wu, J. (2012). Cluster Analysis and K-means Clustering: An Introduction. In Advances in K-means Clustering. Springer Theses. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-29807-3_1
[27] Wiskott, L. (2004). Lecture Notes on Principal Component Analysis. Retrieved from https://graphics.stanford.edu/courses/cs233-20-spring/ReferencedPapers/LectureNotes-PCA.pdf
[28] Xu, K., Shen, L., & Liu, L. (2023). Enhancing Column Generation by Reinforcement Learning-Based Hyper-Heuristic for Vehicle Routing and Scheduling Problems. arXiv. https://doi.org/10.48550/arXiv.2310.09686

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY