

Optimising Airline Maintenance Scheduling Decisions

David Torres Sanchez, B.Sc.(Hons.), M.Res



Submitted for the degree of Doctor of
Philosophy at Lancaster University.

May 2020

Abstract

Airline maintenance scheduling (AMS) studies how plans or schedules are constructed to ensure that a fleet is efficiently maintained and that airline operational demands are met. Additionally, such schedules must take into consideration the different regulations airlines are subject to, while minimising maintenance costs. In this thesis, we study different formulations, solution methods, and modelling considerations, for the AMS and related problems to propose two main contributions.

First, we present a new type of multi-objective mixed integer linear programming formulation which challenges traditional time discretisation. Employing the concept of time intervals, we efficiently model the airline maintenance scheduling problem with tail assignment considerations. With a focus on workshop resource allocation and individual aircraft flight operations, and the use of a custom iterative algorithm, we solve large and long-term real-world instances (16000 flights, 529 aircraft, 8 maintenance workshops) in reasonable computational time. Moreover, we provide evidence to suggest, that our framework provides near-optimal solutions, and that inter-airline cooperation is beneficial for workshops.

Second, we propose a new hybrid solution procedure to solve the aircraft recovery

problem. Here, we study how to re-schedule flights and re-assign aircraft to these, to resume airline operations after an unforeseen disruption. We do so while taking operational restrictions into account. Specifically, restrictions on aircraft, maintenance, crew duty, and passenger delay are accounted for. The flexibility of the approach allows for further operational restrictions to be easily introduced. The hybrid solution procedure involves the combination of column generation with learning-based hyperheuristics. The latter, adaptively selects exact or metaheuristic algorithms to generate columns. The five different algorithms implemented, two of which we developed, were collected and released as a Python package (Torres Sanchez, 2020). Findings suggest that the framework produces fast and insightful recovery solutions.

Acknowledgements

I would like thank Dr Burak Boyaci, for our regular meetings and constant stream of exciting ideas, Professor Konstantinos Zografos, and my industrial supervisor, Dr Richard Standing from R² Data Labs (Rolls-Royce), for their patience, guidance, and feedback throughout. I would also like to thank my internal and external examiners, Professors Adam N. Letchford and Julia Bennell for their corrections that improved this thesis.

I thank the EPSRC funded Statistics and Operational Research (STOR-i) Centre for Doctoral Training and Rolls-Royce Holdings plc. I thank everyone involved in STOR-i, students and others, but especially Professors Jonathan Tawn, Idris Eckley and Kevin Glazebrook for their continued support and very useful pieces of advice. STOR-i has managed to create a friendly research environment which has led to me really enjoying my time here.

Last but not least, I thank my family (far and close) for putting up with coffee-fuelled writing mode Dave. Sin vosotros no estaría donde estoy.

Para mis padres.

Declaration

I declare that the work in this thesis has been done by myself and has not been submitted elsewhere for the award of any other degree.

A version of Chapter 3 has been accepted for publication as: Torres Sanchez, D., Boyacı, B., Zografos, K. G. (2020). ‘[An Optimisation Framework for Airline Fleet Maintenance Scheduling with Tail Assignment Considerations](#)’. *Transportation Research Part B: Methodological* **133**, 142 – 164

A version of Chapter 4 is to be submitted for publication as: Torres Sanchez, D., Boyacı, B., Zografos, K. G. (Forthcoming). ‘[A Hybrid Solution Procedure for the Aircraft Recovery Problem with Operational Restrictions](#)’.

A version of Appendix B.3 has been accepted for publication as: Torres Sanchez, D. (2020). ‘[cspy: A Python package with a collection of algorithms for the \(Resource\) Constrained Shortest Path problem](#)’. *Journal of Open Source Software*, **5**(49), 1655.

The word count for this thesis is 42428 words.

David Torres Sanchez

August 2019

Revised – May 2020

Contents

Abstract	I
Acknowledgements	III
Declaration	V
List of Abbreviations	XII
List of Algorithms	XIII
List of Figures	XVI
List of Tables	XIX
1 Introduction	1
1.1 Problem Description	3
1.1.1 Airline Maintenance Scheduling Problem with Tail Assignment Considerations	4
1.1.2 Aircraft Recovery Problem with Multiple Operational Restrictions	7
1.2 Thesis Contents and Contributions	8

2	Extended Literature Review	15
2.1	General Scheduling Problems	15
2.1.1	Classification of Machine Scheduling Problems	16
2.1.2	Classification of Project Planning Problems and the RCPSP	17
2.1.3	Long-term RCPSP formulations	22
2.1.4	A Formulation for the Preemptive RCPSP	29
2.2	Maintenance Scheduling	34
2.2.1	Prognostic-Based Models	35
2.2.2	Modelling Scheduling Problems in Other Industries	48
2.3	Airline Scheduling and Recovery	50
2.3.1	Tail Assignment and Aircraft Recovery	52
2.3.2	Integrated and Semi-Integrated Airline Scheduling and Recovery	59
2.4	Methodology for Chapter 4	73
2.4.1	Column Generation	73
2.4.2	Heuristics	78
2.4.3	Shortest Path Problem with Resource Constraints	83
3	An Optimisation Framework for Airline Fleet Maintenance Scheduling with Tail Assignment Considerations	88
3.1	Introduction	88
3.2	Literature Review	92
3.3	The Proposed Modelling Approach	100
3.3.1	Assumptions	100

3.3.2	Concepts	101
3.3.3	Airline Fleet Maintenance Scheduling with Violations	102
3.3.4	Airline Fleet Maintenance Scheduling with Tail Assignment	110
3.4	Solution Methodology	116
3.4.1	Preprocessing Routine	116
3.4.2	Solution Procedure	118
3.5	Model Application and Computational Tests	125
3.5.1	Single Workshop Case	127
3.5.2	Multi-workshop Case	134
3.6	Conclusions	140

**4 A Hybrid Solution Procedure for the Aircraft Recovery Problem
with Operational Restrictions 143**

4.1	Introduction	143
4.2	Literature Review	146
4.3	Modelling	156
4.3.1	Definitions	156
4.3.2	Aircraft Recovery Problem Formulation	157
4.3.3	Column Generation	157
4.4	Solution Methodology	166
4.4.1	Network Generation	166
4.4.2	Hybrid Solution Procedure	168
4.4.3	Solving the Subproblems	171

<i>CONTENTS</i>	IX
4.5 Computational Experiments	184
4.5.1 Analysis of Hyper-heuristics	186
4.5.2 Recovery	190
4.6 Conclusions	193
5 Thesis Conclusions, Limitations and Further Work	196
A Appendix for Chapter 3	201
A.1 Extension: Airline Maintenance Scheduling with Flight Re-Scheduling	201
A.2 Supplementary Proofs	202
A.3 Interval Generation	204
B Appendix for Chapter 4	209
B.1 Functions Employed in Metaheuristic Algorithms	209
B.1.1 Algorithm 5	209
B.1.2 Algorithms 6 and 7	210
B.1.3 Algorithm 8	212
B.2 Parameters in the Computational Tests	213
B.3 cspy: A Python package with a collection of algorithms for the (Re-	
source) Constrained Shortest Path problem	215
B.3.1 Introduction	215
B.3.2 Algorithms	216
B.3.3 Features	217
B.3.4 Examples	218

CONTENTS

X

Bibliography

219

List of Abbreviations

AMS : Airline Maintenance Scheduling

AR : Aircraft Recovery

CBM : Condition-based Maintenance

CM : Corrective Maintenance

FH : Flying Hours

GRASP : Greedy Randomised Adaptive Search Procedure

IP : Integer Program

LP : Linear Program

MCNF : Multi-Commodity Network Flow

MIP : Mixed Integer Program

MMILP : Multi-Objective Mixed Integer Linear Program

MOPs : Maintenance Opportunities

MR : Maintenance Requirement

PM : Preventive Maintenance

PRCPSP : Preemptive Resource Constrained Project Scheduling Problem

RCPSP : Resource Constrained Project Scheduling Problem

REF : Resource Extension Function

RR : Rolls-Royce

RUL : Remaining Useful Life

SPPRC: Shortest Path Problem with Resource Constraints

TA : Tail Assignment

TSN : Time-Space Network

List of Algorithms

1	Column Generation Algorithm.	78
2	Generic monodirectional labelling algorithm for the SPPRC.	87
3	Solution procedure with conflicting period selection and interval splitting.	124
4	Diving with LDS (Sadykov, Vanderbeck, Pessoa, Tahiri and Uchoa, 2019).	171
5	Bidirectional labelling algorithm with dynamic halfway point for the SPPRC (Tilk et al., 2017).	176
6	Tabu Search for the SPPRC.	179
7	Greedy Elimination Algorithm for the SPPRC.	180
8	Adapted GRASP for the SPPRC (Ferone et al., 2019).	182
9	Reinforcement Learning – Great-Deluge Hyper-heuristic (Özcan et al., 2010)	183

List of Figures

1.1	Connections between areas in Chapter 2 and the rest of the thesis. . .	9
2.1	An activity broken down into two subactivities.	30
2.2	Example of a feasible preemptive schedule showing the processing of activities for two resources R_1 (left) and R_2 (right).	33
2.3	Example of a feasible non-preemptive schedule showing the processing of activities for two resources R_1 (left) and R_2 (right).	34
2.4	Failure rate (r) evolving through time (t) after performing PM interventions at times t_i for $i = 1, 2, 3, 4$ (assuming $\lambda \geq 1$).	37
2.5	Sequential modelling of the airline scheduling problem (Bae, 2010). .	51
2.6	Hyper-heuristic algorithm.	83
3.1	Stages of the airline scheduling problem.	94
3.2	Time-Space Network (TSN) example.	95
3.3	Timeline showing the deconstruction of three MOPs into sets of consecutive intervals.	104
3.4	Flow chart outlining the process of the iterative algorithm.	117

3.5	Conflicting period selection.	119
3.6	Interval splitting stage for a 9 hour MOP using binary segmentation for three iterations.	122
3.7	Resource profiles for the Bangkok workshop for the first and last (4th) iterations of the interval splitting stage using Method 3.	134
3.8	Result comparison (objectives 3.3.21, 3.3.22) for Method 3 vs tradi- tional discretisation for different time steps.	135
3.9	Computational comparison for Method 3 applied to different multi- workshop cases.	137
3.10	Maintenance schedules produced by using Method 3 for each aircraft and workshop in the first multi-workshop case.	139
4.1	Time-Space Network (TSN) example (Torres Sanchez et al., n.d.). . .	148
4.2	Generation of Time-Space Networks.	168
4.3	Hybrid Solution Procedure.	170
4.4	Effect of two types of disruption on instance I1, airport (left) and air- craft (right).	192
4.5	Effect of two types of disruption on instance I2, airport (left) and air- craft (right).	192
4.6	Effect of two types of disruption on instance I3, airport (left) and air- craft (right).	192
4.7	Effect of two types of disruption on instance I4, airport (left) and air- craft (right).	192

4.8	Effect of two types of disruption on instance I5, airport (left) and aircraft (right).	192
A.1	Example of edge labelling for flights between airports s_p and s_q	205
A.2	Incident edges with source and sink functions.	205

List of Tables

3.1	Typical maintenance frequencies in calendar months (MO), flying hours (FH), or flight cycles (FC) (Cook and Tanner, 2008; Martins, 2016).	90
3.2	Summary of selected literature. MS = inclusion of maintenance scheduling, FH = use of legal remaining flying hours, RC = workshop resource considerations, H = time horizon (days), Sched. Type = scheduling type, Airlines = number of airlines considered, W = number of workshops considered, F = number of flights (in largest test instance), A = number of aircraft (in largest test instance), T = computational time (of largest test instance).	99
3.3	Maintenance regulation parameter values for two check types (1 and 2) and two aircraft types (A320 and F100).	126
3.4	Sample resource demands and limit capacities for four types of resources (r_i for $i = 1, 2, 3, 4$) and two maintenance types (1 and 2).	126
3.5	Objective function breakdown for the Bangkok workshop. The best method in each iteration is highlighted in green.	132

3.6	Comparison across 5 workshops during the interval splitting stage for the three different splitting methods.	133
3.7	Method comparison for the 4-workshop case.	136
4.1	Flight range according to flight length (Eurocontrol, 2011).	144
4.2	Summary of selected AR literature. Air. = aircraft type flight range considerations, Maint. = maintenance regulations considerations, Pass. = passenger delay restrictions considerations, SPPRC = use of shortest path problem with resource constraints, HH = use of hyper-heuristics, Opt. = close-to-optimal integer solution obtained, F = number of flights in largest test instance, A = number of aircraft in largest test instance, T = computational time of largest test instance.	155
4.3	Test instances. ¹ F = # of flights, A = # of aircraft, T = # of aircraft types, H = length of recovery horizon in days, TSN= size of time-space network (# of nodes, # of arcs).	185
4.4	Algorithms used in solution procedure.	186
4.5	Usage count of metaheuristics for the two hyperheuristics (HH1, HH2) using different utility functions.	187
4.6	Computational results of the hybrid solution procedure using hyper-heuristic HH1.	189
4.7	Computational results of the hybrid solution procedure using hyper-heuristic HH2.	189

B.1	Different aircraft types in test instances with their respective operating costs.	214
B.2	Parameters for computational tests.	214

Chapter 1

Introduction

The industrial partner for this PhD project, Rolls-Royce Holdings plc (RR), is a market leader in the civil aerospace business sector; powering more than 13,000 engines around the world and 35 types of commercial aircraft (Rolls-Royce Holdings plc, 2015). Besides producing jet engines, RR provides several so-called **digital service solutions** that range from the detection of operational deficiencies of the engines to the suggestion of tentative maintenance schedules for the management of an airlines' fleet. RR has to cater the appropriate support for all stages of an engines' lifecycle to meet with the service offering requested by the customer. In particular, the services related to this research project are those linked to the scheduling of maintenance for which current services have been found to yield suboptimal plans that induce significant losses. The objectives of this thesis are, to develop optimisation-based mathematical models and solution algorithms to optimise the aircraft maintenance schedules, and, additionally, to provide innovative airline scheduling tools to broaden the digital service solutions offering.

Motivated by these objectives, research has been carried out around several areas of the literature. One such stream is the **airline maintenance scheduling** (AMS) literature. Due to the constant interaction between aircraft maintenance activities and airline operations, a facet of particular interest lies in the interaction between the currently disjoint AMS and **tail assignment** (TA) problems. The latter, refers to the assignment of aircraft to different, already scheduled, flight legs. The two individual components of the problem are of interest to researchers, as they each compose highly complex optimisation problems. Additionally, sets of constraints and regulations have to be included. Typically, these problems are treated separately and solved sequentially (Cordeau et al., 2001). That is, the optimal solution from the TA problem becomes input for the AMS problem. Clearly, the solution of one subproblem does not take into consideration the restrictions of the subsequent subproblem, leading to suboptimal overall solutions, often far from optimality. Nonetheless, considering them in conjunction (keeping in mind tractability), captures the interactions between the two problems, avoids the suboptimality issue, and, maximises aircraft availability while minimising unnecessary maintenance costs and meeting minimum safety regulations as established by the regulatory agencies (civil aviation authority, federal aviation administration or similar).

Another area of interest is the **aircraft recovery** (AR) literature. With maintenance and other operational restrictions in mind, the aim of the AR problem is to recover airline operations from an unexpected disruption. This involves the generation of new flight schedules such that the alterations to the initial plans are minimised.

1.1 Problem Description

The increasing customer demand for *care packages* plus other services, which make up for 53% of the total underlying revenue in RR's civil aerospace business sector, has driven to an exploration and revision of the current systems in place (Rolls-Royce Holdings plc, 2015). The combination of AMS and TA will lead to optimal maintenance decisions which RR believe can significantly increase their revenue aftermarket growth. As for their customers, it is estimated that around 69% of the direct operating costs are engine influenced costs (Rolls-Royce Holdings plc, 2016). Hence, this approach will have a great financial impact on customers as well, not only for the aircraft availability but also in the long-term effect of efficient aircraft maintenance.

In an increasingly larger aviation intelligence market, a brand-new AR learning-based service is bound to have an impact (IATA, 2019). Striving towards one of RR's aims to "Take Care for our customers to the next level" by strengthening their aerospace digital service solutions offering.

In this thesis, we present two new scheduling tools that enable customers to access, maintenance scheduling and workshop services, through our AMS tool; and, aircraft recovery solutions, through our AR learning-based tool.

The individual characteristics of the problems were gathered across several meetings with Dr Richard Standing and other experts from RR including: fuel consultants, the principal business analyst, and past airline practitioners. They can be summarised as two problems with different characteristics. The problems are discussed in the next two sections.

1.1.1 Airline Maintenance Scheduling Problem with Tail Assignment Considerations

The first problem considered in this thesis, subject of Chapter 3, considers the AMS problem with TA considerations and can be labelled as **large-scale**, **medium-term**, **time-dependent** and **prognostic**.

The problem is **large-scale** due to the multiple aspects. Aside from the time dimension, another factor of significant influence is the fleet size and characteristics. Each aircraft belongs to a certain fleet type, hence, has different specifications and passenger restrictions which limit the authorised flight legs. Additionally, each aircraft has a corresponding set of operations history (e.g. the number of flying hours, the number of operations, maintenance details) that need to be considered. Maintenance workshop constraints also introduce further size considerations. Such workshop constraints include, capacity, costs, demand, manpower, and inventory. Notably, manpower or maintenance crew introduces several complications. Restrictions apply to each technician as they can only perform operations on the levels they have been certified for.

The planning horizon for the problem is a **medium-term** 1-3 months. The reason for the planning horizon to have been modified from the usual operational level found in the literature (1 to 7 days), is to capture the interactions between the components of the model and to gain insights into which tactical decisions should be made. Such tactical decisions will favour an overall profit in the long-term by guiding the decisions towards the exploration of alternatives and co-operation options.

The **time-dependent** element, sometimes referred to as dynamicity, comes naturally from the scheduling setting. In optimisation problems, time-dependence can be modelled using either a discrete or a continuous time representation. These choices bring different benefits. As can be found in the reviews of scheduling approaches for chemical processes, Floudas and Lin (2004, 2005) among others, it is apparent that traditional continuous time representations provide higher quality solutions, however, they are more computationally expensive (to the extent that they become intractable for large instances). Since our problem is large, a time discretisation method is employed. In order to improve the quality of the solutions and avoid suboptimality issues, an appropriate solution algorithm is developed that makes the discretisation more granular.

The trade-off between performing different types of interventions at certain workshops and the outcome in terms of improved aircraft condition, cost and time required, can be included using **prognostic** modelling. This allows the up-to-date “health”-state of the aircraft and deterioration to be utilised when making scheduling decisions. The updated legal remaining flying hours can be used to account for aircraft “health”-state. Hence, aircraft have to be monitored throughout the planning horizon to determine whether they are suitable for operating a certain flight, or not; in which case the aircraft can be exchanged for a suitable one. Due to the previous aspects and to preserve tractability, in this thesis, such deterioration does not take into account stochastic considerations such as unexpected breakdowns and maintenance calls. Therefore, this problem focuses on deterministic planning of maintenance.

Current Industrial Practice

Aircraft maintenance requirements are covered by civil aviation authorities. In short, they provide a set of limitations on aircraft use between different types of maintenance interventions. Not considering short-term types, maintenance interventions consist of four medium/long-term airframe checks (A, B, C and D) and three long-term off-wing/major maintenance (engine, landing gear, and auxiliary power unit). The limitations imposed by regulations are a certain number of flying hours, cycles, or months between different maintenance interventions. These vary depending on the aircraft type (Cook and Tanner, 2008). For more information see Section 3.1.

In practice, according to practitioners, maintenance operators constrain themselves by time, not by scope. That is, if an aircraft has been brought in for a non-major check, type A, for example, they will perform as many operations as possible to make the aircraft available by the end of the set downtime. If they are unable to perform all the desirable tasks but the aircraft is safely functional, they will add these to the list of deferred operations to avoid introducing any further delays into the system. The operations in the list will be fixed at a later stage or workshop. For our purposes, two maintenance intervention types will be used, on-wing, check types A and C. Check type B is excluded as it is running out of practice (Qantas, 2016).

For the AMS problem, RR currently employ two pieces of software: **SCAF** and **StaggerLite**, which can be described as a forecaster and a schedule planner, respectively (Rolls-Royce Holdings plc, 2017). The **SCAF** model employs AnyLogic, a customisable simulation tool, to predict engine failure rate evolution across time and

usage. To do this, **SCAF** uses relevant data about each aircraft. Some factors that have been found to be influential include: the number of operations, the age of the aircraft, the number of cycles (takeoffs and landings), and the number of flying hours. The output of the **SCAF** model is then fed into the **StaggerLite** model which, using the **OptiQuest** optimisation tool, arranges a feasible maintenance schedule according to several maintenance requirements, available time slots and objective function criteria. The total computational time for an average size airline ranges from 40 minutes to 1 hour to produce a 2 to 4 week schedule. Unfortunately, this procedure leads to suboptimal schedules which are then not presented in an interpretable way to the maintenance operators and engineers. Moreover, it does not take into account the effect that these interventions may have on TA.

1.1.2 Aircraft Recovery Problem with Multiple Operational Restrictions

The second problem considered in this thesis, subject of Chapter 4, considers the AR problem with operational considerations and can be labelled as **combinatorial**, **short-term**, and **learning-based**.

The nature of the problem is **combinatorial**. Generating new flight schedules around a given disruption, involves selecting from all the possible flight paths. We use column generation to solve the problem efficiently. Moreover, considering multiple operational restrictions reduces the number of feasible flight paths. Among the most important operational restrictions are: aircraft type, maintenance, crew duty,

and passenger delay. Aircraft type restrictions ensure that aircraft operate suitable flight ranges according to their specifications. Maintenance restrictions, as previously discussed, involve limiting the number of flying hours between certified maintenance interventions. Crew duty restrictions involve several rules including enforced rest periods and maximum duty durations. Passenger delay restrictions set maximum delay thresholds after which passengers have the right to reparations by the airline. For more details on operational restrictions, see Section 4.1.

The problem is **short-term** or operational. The aim of the problem is to resume airline operations in minimal time while also minimising the inconveniences for crew and passengers. Hence, an operational planning horizon (1 to 10 days) is appropriate and commonly used for this type of problem.

To provide an aviation intelligence solution, the problem is **learning-based**. For this purpose, the exploration of an interaction between statistics and operational research is required. To incorporate a learning-based mechanism into a column generation setting, selection hyper-heuristics can be employed. These choose different algorithms to generate columns, which can then be evaluated and learning can be done by comparing the performance and quality of the columns.

1.2 Thesis Contents and Contributions

Chapter 2, includes an extended literature review, with topics of interest for Chapters 3 and 4. Figure 1.1 summarises the links and overlaps between the different sections in Chapter 2 and their use in the rest of the thesis. Hence, justifying the areas visited

by either a direct link or an important theoretical relation.

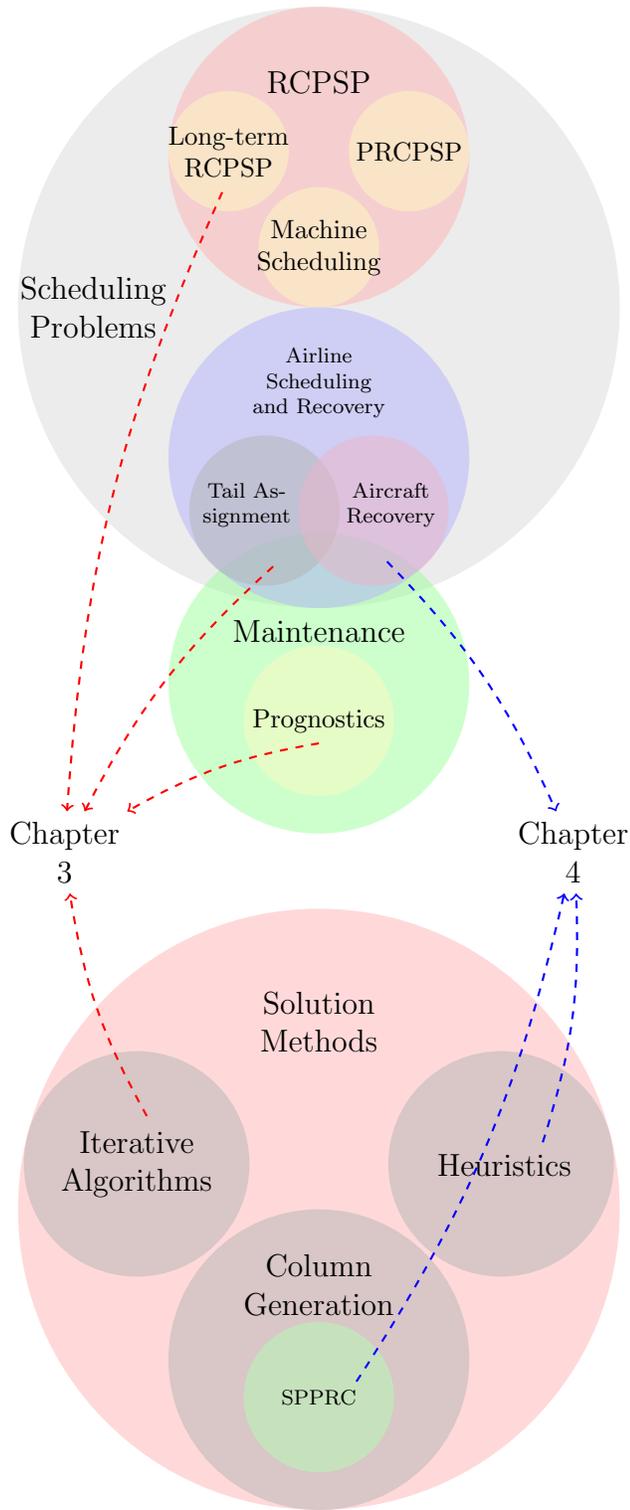


Figure 1.1: Connections between areas in Chapter 2 and the rest of the thesis.

The extended literature review in **Chapter 2**, starts with an introduction to the general scheduling problem, Section 2.1 (Traditional Scheduling Problems), which includes a classification scheme for different types of machine and project scheduling problems. Section 2.1.2 reviews a traditional formulation of the resource constrained project scheduling problem (RCPSP), a special kind of project scheduling which generalises some of the machine scheduling problems. Section 2.1.3 reviews some more recent formulations for the long-term RCPSP. Also, as a minor contribution, Section 2.1.4 presents a new formulation for the preemptive RCPSP (PRCPSP).

The following sections review two streams of the literature, Maintenance Scheduling and Prognostics (Section 2.2), and Airline Scheduling and Recovery (Section 2.3). As can be seen in Figure 1.1, Maintenance Scheduling lies partly inside Scheduling Problems, intersecting with Airline Scheduling, and, partly outside, where prognostic considerations are. Airline Scheduling and Recovery, which contains Tail Assignment and Aircraft Recovery, is fully within Scheduling Problems. Section 2.3.1 examines some well-known formulations for the Tail Assignment and Aircraft Recovery problem. Section 2.3.2 reviews the literature considering intersection between two or more components of the Airline Scheduling and Recovery problems. This section also includes a discussion of different approaches to model the problem such as multi-objective robust optimisation, artificial neural networks, and multi-layered networks.

Section 2.4, reviews several areas related to the methodology employed in Chapter 4. Precisely, Section 2.4.1 studies a common solution method for combinatorial optimisation problems, Column Generation. Section 2.4.2 provides a brief overview of heuristic methods. Additionally, Section 2.4.3, reviews a related and well-known

subproblem, the shortest path problem with resource constraints (SPPRC) and some of the algorithms to solve it.

Chapter 3, as displayed with red dashed arrows in Figure 1.1, collects several concepts from Chapter 2. Event-based formulations from the Long Term RCPSP literature, reviewed in Section 2.1.3. A deterministic take on Prognostics, reviewed in Section 2.2. A restricted modelling of the Tail Assignment, reviewed in Section 2.3, hence retaining focus on maintenance. Along with, a problem-specific iterative algorithm. Chapter 3 effectively addresses a gap in the airline scheduling literature while presenting a methodological contribution. The abstract for this chapter is given below.

Fierce competition between airlines has led to the need of minimising the operating costs while also ensuring quality of service. Given the large proportion of operating costs dedicated to aircraft maintenance, cooperation between airlines and their respective maintenance provider is paramount. In this research, we propose a framework to develop commercially viable and maintenance feasible flight and maintenance schedules. Such framework involves two multi-objective mixed integer linear programming (MMILP) formulations and an iterative algorithm. The first formulation, the airline fleet maintenance scheduling (AMS) with violations, minimises the number of maintenance regulation violations and the number of not airworthy aircraft; subject to limited workshop resources and current maintenance regulations on individual aircraft flying hours. The second formulation, the AMS with tail assignment (TA) allows aircraft to be assigned to different flights. In this case, subject to similar

constraints as the first formulation, six lexicographically ordered objective functions are minimised. Namely, the number of violations, maximum resource level, number of tail reassignments, number of maintenance interventions, overall resource usage, and the amount of maintenance required by each aircraft at the end of the planning horizon. The iterative algorithm ensures fast computational times while providing good quality solutions. Additionally, by tracking aircraft and using precise flying hours between maintenance opportunities, we ensure that the aircraft are airworthy at all times. Computational tests on real flight schedules over a 30-day planning horizon show that even with multiple airlines and workshops (16000 flights, 529 aircraft, 8 maintenance workshops) our solution approach can construct near-optimal maintenance schedules within minutes.

On the other hand, **Chapter 4**, as shown with blue dashed arrows in Figure 1.1, draws upon various concepts from Chapter 2. The modelling of the Aircraft Recovery using a set covering/packing formulation, reviewed in Section 2.3, with maintenance and several other operational considerations (with maintenance not being the primary focus). The use of Column Generation discussed in Section 2.4.1; The use of hyper-heuristics, discussed in Section 2.4.2, in the column generation context. Also, the use of SPPRC to model subproblems and the use, development, and improvement of different exact and metaheuristic algorithms to solve them, discussed in Section 2.4.3. Chapter 4 presents a methodological contribution. The abstract for this chapter is given below.

The unforeseen disruption of airline services often leads to significant additional

costs. Aircraft recovery models enable the resumption of services in minimal time while minimising the additional costs incurred. Therefore, solutions must provide an efficient re-planning of scheduled services that mitigates the impact on passengers and crew while respecting a set of regulations for operations. Such regulations impose restrictions on aircraft, maintenance, crew, and passenger delay. The aircraft recovery problem is a hard combinatorial optimisation problem, hence, these restrictions cannot be modelled with ease. In this paper, we develop a hybrid solution procedure, based on column generation and hyper-heuristics, that allows us to consider such operational restrictions while providing solutions to real-world test instances in reasonable computational time. Such solutions, minimise the number of flights that are cancelled and the total operating costs. The operational restrictions are taken into account in the subproblems, i.e. when generating columns. Specifically, this is done by modelling the subproblems as shortest path problems with resource constraints, with appropriately defined weights and resources, one for each operational restriction. The hyper-heuristic algorithms learns from the selection of three different metaheuristic algorithms to generate columns. The learning process allows the hyper-heuristic to adapt as the problem develops, selecting the most effective metaheuristic for each subproblem. Furthermore, an exact bidirectional labelling algorithm with dynamic halfway point is used to generate columns with non-negative reduced costs, thus, guaranteeing optimal solutions for the LP relaxation of the aircraft recovery problem. To obtain close-to-optimal integer solutions to the aircraft recovery problem while preserving computational efficiency, we employ a diving heuristic as part of our hybrid solution procedure. Additionally, for the implementation of this work, we employ the

open-source Cbc solver ([johnjforrest et al. 2020](#)).

Lastly, **Chapter 5** provides the concluding remarks for the thesis and potential research directions. Additionally, Appendix A provides supplementary materials for Chapter 3, an extension that accounts for flight re-scheduling, some additional proofs, and the theoretical motivation behind the interval-based formulations. Appendix B provides extra materials for Chapter 4, functions used in the pseudo-code for each algorithm, parameters used in the computational tests, and the high-level paper about the Python package developed.

Chapter 2

Extended Literature Review

This chapter contains a review of the literature necessary for Chapters 3 and 4. Particularly, we include an introduction to **general scheduling problems** (machine scheduling and project planning), an overview of **maintenance scheduling and prognostics**, an exploration of **airline scheduling and recovery**, and a breakdown of the areas employed in the **methodology for Chapter 4**.

2.1 General Scheduling Problems

Since the late '50s, scheduling problems have been researched thoroughly. There were mainly developed and motivated around two applications: **machine scheduling** and **project planning**. Machine scheduling deals with the combinatorial problem that arises from considering a large number of scheduling situations in which one or more machines are available for processing a particular number of jobs. Project planning refers to the programming of different activities that need completion for

a given project. Various types of projects are, for example, a school timetable, a construction, a business expansion, or a software development process. These usually are accompanied by some precedence and resource constraints. Such resources can be broadly defined as “production units” and range from raw materials to machinery or even manpower and are taken as inputs for the project planning problem. Brucker and Knust (2012) gave a very comprehensive study to both of these problems.

2.1.1 Classification of Machine Scheduling Problems

Depending on the machine environment or on the characteristics of the jobs there are several classes of machine scheduling problems. As it shall be discussed later, these can be generalised under a special type of project scheduling problem.

The simplest machine scheduling model is for a single machine. That is, there are n jobs $\mathcal{J} = \{1, 2, \dots, n\}$ (order of which is assumed to be given), with certain known processing times p_j , and which have to be processed on a single machine. Under the assumption that only one job can be processed at a time, this case is known as the **single machine scheduling** problem. The next simplest case is when a set of jobs have to be processed on multiple independent and identical machines with the same processing times. However, if the machines are not identical, then the processing time for job j on machine M_k becomes p_{jk} . If a machine is allowed to process more than one different job, then any machine in the subset of machines $\mu_j \in \{M_1, \dots, M_m\}$ can be used to process it, unless that machine is being used to process another job. All of these cases (subject, again, to the single job per machine assumption) are grouped under the **parallel machine scheduling** problem. More precisely, when jobs occupy

all machines in their respective μ_j simultaneously, then the problem is classed as a **multi-processor task scheduling** problem.

By studying the characteristics of different jobs, more classes of machine scheduling problems arise; **shop scheduling** problems. In shop scheduling problems, jobs consist of a sequence of operations that have to be processed on different machines. Formally, job j consists of n_j operations O_{1j}, \dots, O_{n_jj} ; that must be processed sequentially (in some order) and in one machines from the subset, $\mu_{O_{ij}} \in \{M_1, \dots, M_u\}$ (for $1 \leq u \leq m$ and $1 \leq i \leq n_j$). If there is a precedence relation between operations but jobs are independent of each other, then the problem is classed as a **job-shop** problem. If there is an order for both, the operations and the jobs, then the problem becomes a **flow-shop** problem. Further, if there is an order for the jobs but not for the operations, then the problem is classed as a **open-shop** problem.

2.1.2 Classification of Project Planning Problems and the RCPSP

Project planning is heavily conditioned by the specifications on the resources and activities. These are grouped under the more appropriate term, **resource-constrained project scheduling problem** (RCPSP). For a recent and thorough review, we refer the reader to Habibi et al. (2018). The generality of the RCPSP allows it to have a wide range of applications where the aim is to schedule some activities or jobs, such that precedence and resource constraints are satisfied, and a certain objective function is optimised. As for the objective functions, for example, the project duration, the

deviation from deadlines, or costs concerning resources, may be minimised.

To study the RCPSP, let us introduce some notation. Let T denote a time horizon and $t \in \mathcal{T} = \{0, 1, \dots, T\}$ denote a discrete time unit. There are n activities belonging to the set \mathcal{J} (as previously defined), and a set of r resources $\mathcal{R} = \{1, 2, \dots, R\}$. For every activity $j \in \mathcal{J}$, there is an associated cost c_j , duration p_j , deadline $d_j \in \mathcal{T}$, and resource requirements r_{jk} (for each $k \in \mathcal{R}$). Also, the completion time of an activity C_j , is directly related to the starting time $S_j \in \mathcal{T}$, by the equation $C_j = S_j + p_j$.

Let the set of precedence relations be denoted by E . Precedence constraints for activities are defined as follows. If activity j cannot begin until activity i is fully completed, then $S_i + p_i = C_i \leq S_j$, and we write $(i, j) \in E$. This is called a **finish-to-start** relation and it can be relaxed by allowing activity j to start before activity i is fully completed (start-to-start relation). Using a **minimum time-lag** l_{ij} , the constraint becomes $C_i + l_{ij} \leq S_j$, and it ensures that the time between the completion time of activity i and the starting time of j is at least l_{ij} .

Resource constraints depend on the availability of resources. Resource availability can be of different types: renewable, nonrenewable, partially renewable, or cumulative; each of these requires different modelling. By setting the total resource demand less than or equal to R_k for each resource k at any time point, the **renewable availability** constraint will be formulated. Perhaps a more realistic case is the one with a **cumulative availability** or storage resources. This case allows the resource availability at a particular time to depend on the resource requirement of activities that have already been performed. Cumulative availability can also be used to model a resource that is available in storage and can be depleted or replenished conveniently,

perhaps incurring additional costs. Let resource demands r_{jk}^- and r_{jk}^+ , represent the depleted and replenished amount, respectively, of resource k when activity j is performed. By introducing a dummy starting activity 0 and dummy terminating activity $n + 1$ the initial and final amount of resource k can be modelled by r_{0k}^+ and $r_{n+1,k}^+$, respectively. For each resource k , there is a lower bound (safety stock) \underline{R}_k and upper bound (maximum capacity) \overline{R}_k . The constraints to ensure the accumulated inventory for every resource stays within its bounds is given by,

$$\underline{R}_k \leq \sum_{\{j \mid S_j + p_j \leq t\}} r_{jk}^+ - \sum_{\{j \mid S_j \leq t\}} r_{jk}^- \leq \overline{R}_k, \quad \forall t \in \mathcal{T}, k \in \mathcal{R}.$$

Furthermore, in the case when there are some processing alternatives for activity j (e.g. using more than one machine to process it), the problem becomes **multi-mode**. For this extension, processing alternatives may be grouped in the set \mathcal{M}_j of modes (conceptually the same as the μ_j seen in the machine scheduling problem) and the processing time and resource usage of activity j will depend on the mode $m \in \mathcal{M}_j$. These can be denoted by p_{jm} and r_{jkm} , respectively.

Formulation of the RCPSP

After studying the necessary considerations and classifications of the problem, a basic formulation can be presented. For the sake of illustration, a RCPSP model with minimum time-lag, renewable resources, and a generic cost function is presented. Let x_{jt} be the binary decision variable for this problem; it is equal to 1 if activity j starts at time period t , and it is 0 otherwise. The RCPSP can be formulated as follows,

Model 2.1.1. *A discrete-time (DT) formulation for the RCPSP (Brucker and Knust, 2012)*

$$\min_x \sum_{t \in \mathcal{T}} \sum_{j \in \mathcal{J}} c_j(t) x_{jt} \quad (2.1.1a)$$

Subject to

$$\sum_{t \in \mathcal{T}} x_{jt} = 1, \quad \forall j \in \mathcal{J} \quad (2.1.1b)$$

$$\underbrace{\sum_{t \in \mathcal{T}} t x_{it} + p_i + l_{ij}}_{C_i} \leq \underbrace{\sum_{t \in \mathcal{T}} t x_{jt}}_{S_j} \quad \forall (i, j) \in E \quad (2.1.1c)$$

$$\sum_{j \in \mathcal{J}} \sum_{u=t-p_i+1}^t r_{jk} x_{ju} \leq R_k \quad \forall k \in \mathcal{R}, t \in \mathcal{T} \quad (2.1.1d)$$

$$x_{jt} \in \{0, 1\} \quad \forall j \in \mathcal{J}, t \in \mathcal{T} \quad (2.1.1e)$$

The generic time-dependent cost function $c_j(t)$, used in the objective function 2.1.1a, can take any of the following forms,

- (i) **Makespan:** to minimise the total project time, we can set $c_j(t) = 0$ for all activities $j \in \mathcal{J} \setminus \{n+1\}$, and $c_{n+1}(t) = t$;
- (ii) **Maximum lateness:** to minimise the maximum difference between the completion time and the deadline, we can use the same $c_j(t)$ as for objective (i), and additionally, set $l_{j,n+1} = p_j - d_j$ for all activities $j \in \mathcal{J}$;
- (iii) **Earliness/tardiness costs:** to minimise costs associated with early/late activities, we can set $c_j(t) = \alpha_j \max\{0, d_j - C_j\} + \beta_j \max\{0, C_j - d_j\}$, where α_j and β_j are the additional costs associated with earliness and tardiness for each activity j ;
- (iv) **Weighted number of late activities:** to minimise the weighted number of late activities, we can use the same cost function as for objective (iii) with

$$\alpha_j = 0.$$

Constraints 2.1.1b specify that only one start time is allowed for every activity. Precedence constraints 2.1.1c ensure that finish-to-start precedence relations with minimum time-lag are enforced (recall $C_i + l_{ij} \leq S_j$). More precisely, for any pair of activities in the precedence set $(i, j) \in E$, the time between the completion time of activity i (given by the first sum plus the processing time) and the starting time of its successor j (given by the LHS), must be at least the minimum time-lag l_{ij} . Resource constraints 2.1.1d guarantee that the capacity of the resources is not exceeded for the duration of the activity. This integer program (IP) has nT variables, $|E|$ precedence constraints captured in constraints 2.1.1c, and RT resource constraints 2.1.1d.

A variant of this formulation, with stronger precedence constraints, is given by the disaggregated discrete-time (DDT) formulation. To achieve this, we can replace constraints 2.1.1c with the following,

$$\sum_{u=t}^T x_{iu} + \sum_{u=0}^{t+p_i+l_{ij}-1} x_{ju} \leq 1 \quad \forall (i, j) \in E, t \in \mathcal{T} \setminus \{T-1\}. \quad (2.1.2)$$

These constraints ensure that if activity i starts at time t , then activity j cannot start until time $t + p_i + l_{ij}$, hence, enforcing the required finish-to-start precedence relation with minimum time-lag. The reason why the DDT formulation is stronger than the DT formulation is because constraints 2.1.1b and 2.1.2 imply constraints 2.1.1c. Hence, when computing lower bounds using the linear relaxation of both formulations, the DDT provides better lower bounds. However, due to the higher number of constraints, this leads to longer computational times (Brucker and Knust,

2012).

As previously mentioned, machine scheduling problems are special cases of the RCPSP. For example, the parallel machine scheduling problems with non-identical machines can be modelled as a multi-mode RCPSP with R renewable resources and $R_k = 1$ for $k \in \mathcal{R}$. Similarly, the general-shop scheduling problem can be modelled as the RCPSP with $m + n$ renewable resources, $R_k = 1$ for all resources k . Of these, the first m correspond to the machines and the remaining resources account for the fact that different operations of job j cannot be processed at the same time. The total number of activities will be $\sum_{j=1}^n n_j$ and operation O_{ij} requires one unit of machine resource μ_{ij} and one unit of job resource $m + j$.

2.1.3 Long-term RCPSP formulations

Given the nature the problem considered in Chapter 3, the long-term RCPSP literature is worth considering. Particularly, Koné et al. (2011) introduced a formulation that challenged the classical discretisation of time by indexing variables with some pre-defined time intervals. They refer to this type of formulation as an “event-based RCPSP”. More formally, these events correspond to start or end times of activities. Their formulation, for long-term planning horizons, involves considerably fewer variables than the formulations indexed by time. It should be noted that they independently developed the same idea as Sousa and Wolsey (1992). Koné et al. (2013) later extended the event-based formulation to account for non-renewable resources. More authors have used a similar indexation of time. Naber (2017) removed the assumption regarding fixed resources per activity in the RCPSP by allowing flexible

resource usage. Kopanos et al. (2014) presented two new types of continuous time formulations and provided a computational study comparing the performance of other similar RCPSP formulations.

On/Off Event-based Formulation (Koné et al., 2011)

To study the “event-based RCPSP” formulations by Koné et al. (2011), let us alter the notation used in the previous section to match the one employed by the authors. Let the RCPSP be defined by (A, p, E, R, B, b) . Where A is a set of activities (including dummy start and end activities, i.e. $A = \{0, 1, \dots, n, n+1\}$), p is a vector of durations, E is the precedence relation, R is a set of renewable resources, B is a vector of resource availabilities, and b is a matrix of demands. For the event-based formulation we require a set of events $\mathcal{E} = \{0, 1, \dots, n\}$, where n is the number of activities. Each of these correspond to either the start or the end of an activity. Making use of this event set Koné et al. (2011) introduced two event-indexed formulations for the RCPSP. The first formulation, the Start/End event-based (SEE) formulation, is a simplification of the formulation by Zapata et al. (2008) for the multi-mode multi-project RCPSP. The SEE formulation requires two variables for each event, which allows tracking of the start and end events for each activity. The formulation we are going to discuss in more detail is the On/Off event-based (OOE) formulation as it only requires one variable per event. It receives its name due to the definition of the variables used as they either represent the start or the end of an activity. With this definition it is more difficult to identify when an activity starts and ends; for this purpose, the authors include some “contiguity” (transitivity) constraints and a continuous variable to track

event times. Apart from being able to deal with long-term planning horizons, another advantage of these formulations is that they can deal with rational processing times for activities.

To study the OOE formulation in more detail, let the event indexed binary variable z_{ie} , take the value 1 if an activity i starts at event e or if it is still being processed immediately after event e ; and it is 0 otherwise. The authors introduce a continuous variable t_e in order to account for time/date of event e . In terms of project completion time, the worst case would be if the processing times for all activities is disjoint, meaning that all the t_e variables will be different. However, depending on resource availability, the start/end of different activities may correspond to the same events, which makes some of these times equal. Moreover, the authors work under the following assumptions,

1. Fixed and renewable resources;
2. Non-preemptive schedule;
3. Activities make use of all the resources they require when being processed.

The problem can be formulated as follows,

Model 2.1.2. *OOE formulation Koné et al. (2011).*

$$\min C_{\max} \quad (2.1.3a)$$

Subject to

$$\sum_{e \in \mathcal{E}} z_{ie} \geq 1; \quad \forall i \quad (2.1.3b)$$

$$C_{\max} \geq t_e + (z_{ie} - z_{i,e-1})p_i; \quad \forall i, e; \quad (2.1.3c)$$

$$t_0 = 0; \quad (2.1.3d)$$

$$t_{e+1} \geq t_e; \quad \forall e \neq n-1 \in \mathcal{E} \quad (2.1.3e)$$

$$t_f \geq t_e + ((z_{ie} - z_{i,e-1}) - (z_{if} - z_{i,f-1}) - 1)p_i; \quad \forall (e, f, i) \in \mathcal{E}^2 \times A, f > e \quad (2.1.3f)$$

$$\sum_{e'=0}^{e-1} z_{ie'} \leq e(1 - (z_{ie} - z_{i,e-1})); \quad \forall e \in \mathcal{E} \setminus \{0\} \quad (2.1.3g)$$

$$\sum_{e'=e}^{n-1} z_{ie'} \leq (n - e)(1 + (z_{ie} - z_{i,e-1})); \quad \forall e \in \mathcal{E} \setminus \{0\} \quad (2.1.3h)$$

$$z_{ie} + \sum_{e'=0}^{e-1} z_{je'} \leq 1 + (1 - z_{ie})e; \quad \forall e, \forall (i, j) \in E \quad (2.1.3i)$$

$$\sum_{i=0}^{n-1} b_{ik}z_{ie} \leq B_k; \quad \forall e, k \quad (2.1.3j)$$

$$t_e \geq 0; \quad \forall e; \quad (2.1.3k)$$

$$z_{ie} \in \{0, 1\}; \quad \forall i, e; \quad (2.1.3l)$$

The OOE formulation involves $\mathcal{O}(n^2)$ binary variables, $\mathcal{O}(n)$ continuous variables, and $\mathcal{O}(n^3 + (|R| + |E|)n)$ constraints.

The objective function 2.1.3a, employs a makespan criteria and a corresponding continuous variable subject to constraints 2.1.3c. Constraints 2.1.3b ensure that every activity is processed at least once. Constraints 2.1.3d initialise the time variable for the first event while constraints 2.1.3e ensure that it remains nondecreasing. Constraints 2.1.3f link binary and continuous variables in a form that poses a lower bound for the time of any later event. That is, if activity i starts immediately after event e ($z_{ie} = 1 \wedge z_{i,e-1} = 0$) and ends at event f ($z_{i,f-1} = 1 \wedge z_{i,f} = 0$), then the time of event f

is at least equal to the time of event e plus the processing time of activity i . Any other combination of values for the binary variables yields a redundant constraint. Constraints 2.1.3g and 2.1.3h are called contiguity constraints, more commonly referred to as transitivity, and ensure that z_{ie} remains equal to 1 for the total processing time of activity i . Constraints 2.1.3i ensure precedence relations are satisfied. Constraints 2.1.3j ensure that the resource limits are not violated. The last two constraints, 2.1.3k and 2.1.3l define the variables.

If the earliest start (ES_i) and latest start (LS_i) times for activities are known, the following constraints can be included,

$$ES_i z_{ie} \leq t_e \leq LS_i (z_{ie} - z_{i,e-1}) + LS_n (1 - (z_{ie} - z_{i,e-1})) \quad \forall i, e. \quad (2.1.4)$$

These constraints are crucial for the performance of the model, hence, if the earliest/latest start times are not known, preprocessing steps can be implemented in order to compute them (Demasseey et al., 2005).

Koné et al. (2011) included a computational comparison of the OOE formulation against three other well-known formulations. The discrete time formulation, the disaggregated discrete time formulation, and the flow-based continuous-time formulation; as well as their own SEE. Testing on different data sets with different properties clearly showed that the OOE outperforms the other formulations on instance sets with a long time horizon and high processing rates of activities.

RCPSP with consumption and production of resources (Koné et al., 2013)

Koné et al. (2013), extended the OOE formulation (Koné et al., 2011) for a more complex problem; the RCPSP with consumption and production of resources (RCPSP/CPR). They showed that the revised event-based formulation still outperformed all other types. The formulation borrows most of the constraints from Model 2.1.2 but also introduces several continuous variables to account for the stock level with consumption/production of different resources for each activity. To avoid confusion in the notation for the resources we change R to P , as the former previously denoted renewable resources. The continuous variables required are,

s_{ep} : stock level of resource $p \in P$ at event e ;

u_{iep} : amount of resource $p \in P$ consumed by activity i at event e ;

p_{iep} : amount of resource $p \in P$ produced by activity i at event e .

The RCPSP/CPR formulation involves objective 2.1.3a and constraints 2.1.3b to 2.1.3l from Model 2.1.2, as well as some constraints regarding the consumption of resources. The formulation is as follows,

Model 2.1.3. *RCPSP/CPR Koné et al. (2013)*

Constraints 2.1.3a to 2.1.3l

$$u_{iep} \geq c_{ip}^-(z_{ie} - z_{i,e-1}); \quad \forall i, e, p; \quad (2.1.5a)$$

$$u_{iep} \leq c_{ip}^- z_{ie}; \quad \forall i, e, p; \quad (2.1.5b)$$

$$u_{iep} \leq c_{ip}^-(1 - z_{i,e-1}); \quad \forall i, e, p; \quad (2.1.5c)$$

$$p_{iep} \geq c_{ip}^+(z_{i,e-1} - z_{ie}); \quad \forall i, e, p; \quad (2.1.5d)$$

$$p_{iep} \leq c_{ip}^+ z_{i,e-1}; \quad \forall i, e, p; \quad (2.1.5e)$$

$$p_{iep} \leq c_{ip}^+(1 - z_{i,e}); \quad \forall i, e, p; \quad (2.1.5f)$$

$$s_{ep} = s_{e-1,p} + \sum_{i \in A} p_{iep} - \sum_{i \in A} u_{iep}; \quad \forall i, e, p; \quad (2.1.5g)$$

$$\sum_{i=0}^{n-1} b_{ip} z_{ie} \leq s_{ep}; \quad \forall e, p; \quad (2.1.5h)$$

$$s_{ep}, u_{iep}, p_{iep} \geq 0; \quad \forall i, e, p. \quad (2.1.5i)$$

Here, c_{ip}^- and c_{ip}^+ denote the minimum and maximum amount of resource p required to perform activity i , respectively. The combination of constraints 2.1.5a, 2.1.5b, and 2.1.5c ensure that the value of variable u_{iep} is equal to c_{ip}^- if activity i starts being processed at event e for any resource p , and is 0 otherwise. So, the only combination that leads to $u_{iep} = c_{ip}^-$ is $z_{ie} = 1 \wedge z_{i,e-1} = 0$. Similarly, constraints 2.1.5d, 2.1.5e, and 2.1.5f ensure the production variable is equal to c_{ip}^+ if activity i finishes being processed at event e for any resource p , and is 0 otherwise. In this case, the only combination that leads to $p_{iep} = c_{ip}^+$ is $z_{ie} = 0 \wedge z_{i,e-1} = 1$. Constraints 2.1.5g ensure the stock level $s_{e,p}$, for each event e and resource p is updated utilising the stock level from the previous event $s_{e-1,p}$, plus the resource produced by each activity minus the consumption by each activity. Constraints 2.1.5h limit the resources available at each event with the corresponding stock level. Lastly, constraints 2.1.5i define the variables.

2.1.4 A Formulation for the Preemptive RCPSP

The literature for the preemptive RCPSP (PRCPSP) is very limited. Afshar-Nadjafi (2014) studied different preemption penalty costs and a weighted earliness-tardiness in the objective function of the PRCPSP. Afshar-Nadjafi and Arani (2014) published similar results for the multi-mode case. Coffman et al. (2015) explored the smallest time between events (shift) in an optimal schedule for parallel machine scheduling problem with preemptions allowed at non-integer (rational) times. Additionally, in this article, authors introduced an alternative definition for events to account for preemptions, these are identified as an activity start, interruption, resumption, or completion; let us adopt such definition. More recently, Creemers (2019) showed the impact of allowing preemptions in a stochastic project scheduling setting. They presented a computational study (using `psplib` instances Kolisch and Sprecher, 1997) where they compared the benefit in preemption when compared to other well-known RCPSP formulations.

We wish to extend the OOE formulation from Koné et al. (2011) (discussed in Section 2.1.3) to allow preemptions. For this purpose, we require to introduce some notation. Let p_i^- be a value that represents the minimum time allowed to be spent on activity i , with $p_i^- < p_i$ (see Figure 2.1). This can also be thought of as the amount of time after starting/resuming an activity in which we cannot complete/interrupt said activity. From this, the fraction $n_i = \lfloor p_i/p_i^- \rfloor$ determines the maximum number of subactivities activity i can be broken into. Hence,

$$\sum_{i \in A} \left\lfloor \frac{p_i}{p_i^-} \right\rfloor = \sum_{i \in A} n_i = N ,$$

determines the new number of events. That is, $\mathcal{E} = \{0, 1, \dots, N\}$. Furthermore, we impose the assumption, that two subactivities can only be processed in parallel if they do not divide the same activity.

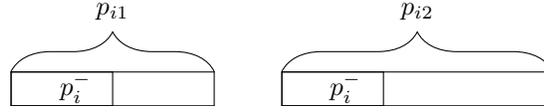


Figure 2.1: An activity broken down into two subactivities.

With these definitions, we can change constraints 2.1.3f to,

$$p_i^- ((z_{ie} - z_{i,e-1}) - (z_{if} - z_{i,f-1}) - 1) \leq t_f - t_e \leq p_i \quad \forall (e, f) \in \mathcal{E}^2 (f > e), \forall i \in A. \quad (2.1.6)$$

These constraints bound elapsed times between events. The upper bound is the total processing time p_i . While the lower bound is set to be the minimum time allowed, p_i^- .

To monitor the progress of individual subactivities throughout the project, let us introduce a new continuous time variable, a_{ie} , which represents the amount of processing time that has been dedicated to activity i by event e . In terms of the variables introduced for Model 2.1.2, for a single process of each activity, we may write the processing variable as follows,

$$a_{ie} = \begin{cases} a_{i,e-1} + t_e - t_{e-1}, & \text{if } z_{ie} = 1 \vee z_{i,e-1} = 1 \\ a_{i,e-1}, & \text{otherwise} \end{cases} \quad \forall i, e.$$

With $0 \leq a_{ie} \leq p_i \forall e \in \mathcal{E}$; $a_{i0} = a_{i,-1} = 0 \forall i \in A$; and $a_{iN} = p_i \forall i \in A$. By using some big- M constraints, this definition is equivalent to the following activity

processing constraints,

$$a_{i0} = a_{i,-1} = 0 \quad \forall i; \quad (2.1.7a)$$

$$a_{i,e-1} \leq a_{i,e} \quad \forall i, e; \quad (2.1.7b)$$

$$a_{ie} \leq a_{i,e-1} + M(z_{ie} + z_{i,e-1}) \quad \forall i, e; \quad (2.1.7c)$$

$$a_{ie} \geq a_{i,e-1} + (t_e - t_{e-1}) - M(1 - z_{ie}) \quad \forall i, e; \quad (2.1.7d)$$

$$a_{ie} \geq a_{i,e-1} + (t_e - t_{e-1}) - M(1 - z_{i,e-1}) \quad \forall i, e; \quad (2.1.7e)$$

$$a_{iN} = p_i \quad \forall i; \quad (2.1.7f)$$

$$0 \leq a_{ie} \leq p_i \quad \forall e \in \mathcal{E} \quad \forall i, e. \quad (2.1.7g)$$

Constraints 2.1.7a initialise the dummy activity processing variable $a_{i,-1}$, and the one corresponding first event a_{i0} . Constraints 2.1.7b ensure that the processing variable is nondecreasing. Constraints 2.1.7c to 2.1.7e ensure that if either $z_{ie} = 1$ or $z_{i,e-1} = 1$ the processing variable is recursively updated, otherwise, due to M , they become redundant. That is, if $z_{ie} = 1 \vee z_{i,e-1} = 1$, a_{ie} is updated with $a_{i,e-1}$, the already processed time for activity i , plus the time elapsed, $t_e - t_{e-1}$, between events $e - 1$ and e . In fact, since constraints 2.1.6 bound $t_e - t_{e-1}$ with p_i , for the same effect we can replace M with p_i . Constraints 2.1.7f ensure that every activity is processed to completion and least by the last event N . Constraints 2.1.7g bound the activity processing variable.

As for the objective function for the event-based PRCPSP we have similar options to those for the simpler version of the problem. In our notation, we have

- (i) **Makespan:** $\min C_{\max} = t_N$.

- (ii) **Maximum lateness:** $\min \max_{i \in A} t_{e(i)} - d_i$, where d_i is the deadline for activity i and $e(i) = \min\{e : a_{ie} = p_i\}$, i.e. the event where the processing for activity i is completed.
- (iii) **Preemption costs:** $\min \sum_i \sum_e c_i^P (z_{i,e-1} p_i - (a_{ie} - a_{i,e-1}))$ where c_i^P is the cost of preemption for activity i .

Choosing one of these objective functions, (i) for instance, and including the constraints from Model 2.1.2, we can present a formulation for the event-based preemptive RCPSP,

Model 2.1.4. *Event-based formulation for the preemptive RCPSP.*

$$\begin{aligned}
 & \min C_{\max} \\
 & \textbf{Subject to} \\
 & \textit{Constraints from Model 2.1.2} \\
 & 2.1.3b \textit{ to } 2.1.3e \\
 & 2.1.6 \\
 & 2.1.3g \textit{ to } 2.1.3l \\
 & \textit{Earliest/Latest start times constraint} \\
 & 2.1.4 \\
 & \textit{Activity Processing Constraints} \\
 & 2.1.7a \textit{ to } 2.1.7f \\
 & \textit{Activity Processing Variable Definition} \\
 & 2.1.7g
 \end{aligned}$$

Example 2.1.1. Consider a preemptive RCPSP with the following properties,

i	1	2	3	4
p_i	4	3	5	8
p_i^-	1	2	2	2
n_i	4	1	2	4
b_{i1}	3	1	2	2
b_{i2}	3	4	2	3

Where resources R_1 and R_2 have capacities 5 and 7 respectively. Also, we have

the following two precedence relations $E = \{(2, 3), (3, 4)\}$ of types finish-to-start and start-to-start, respectively. Meaning that activity 3 cannot start before the end of activity 2, and that activity 4 cannot start before the start of activity 3.

In this case, from the values above, we see that activity 2, for example, has to be processed in a single go ($n_2 = 1$). The others have at most 4 subactivities. Thus, $N = n_1 + n_2 + n_3 + n_4 = 11$. A feasible solution for the scheduling of this project can be seen in Figure 2.2, with the corresponding values for the variables,

$$\begin{aligned}
 e = 0: & \quad t_0 = 0, & \quad z_{20} = 1 & \quad a_{20} = 0 \\
 e = 1: & \quad t_1 = 0, & \quad z_{21} = z_{11} = 1 & \quad a_{21} = a_{11} = 0 \\
 e = 2: & \quad t_2 = 3, & \quad z_{32} = 1 & \quad a_{22} = a_{12} = 3, \quad a_{32} = 0 \\
 e = 3: & \quad t_3 = 3, & \quad z_{33} = z_{43} = 1 & \quad a_{33} = a_{43} = 0 \\
 e = 4: & \quad t_4 = 8, & \quad z_{44} = 1 & \quad a_{34} = a_{44} = 5 \\
 e = 5: & \quad t_5 = 8, & \quad z_{45} = z_{15} = 1 & \quad a_{45} = 5, \quad a_{15} = 3 \\
 e = 6: & \quad t_6 = 9, & \quad z_{46} = 1 & \quad a_{45} = 6, \quad a_{16} = 4 \\
 e = 7: & \quad t_7 = 11, & & \quad a_{47} = 8.
 \end{aligned}$$

With $C_{\max} = 11$. The remaining variables are equal to 0. The Gantt chart of the two resources considered R_1 and R_2 is presented in Figure 2.2. It is worth noting that, in comparison with the non-preemptive case, solution in Figure 2.3, the makespan is reduced by 1.

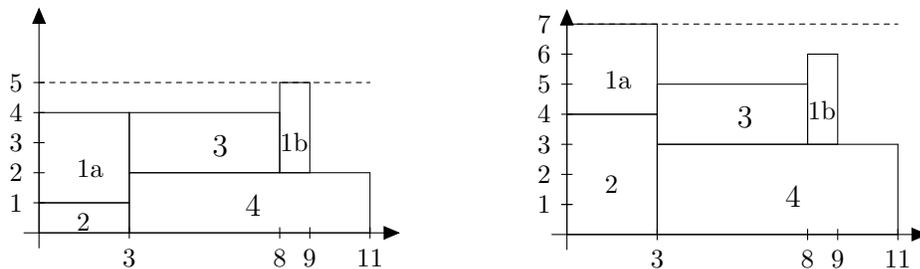


Figure 2.2: Example of a feasible preemptive schedule showing the processing of activities for two resources R_1 (left) and R_2 (right).

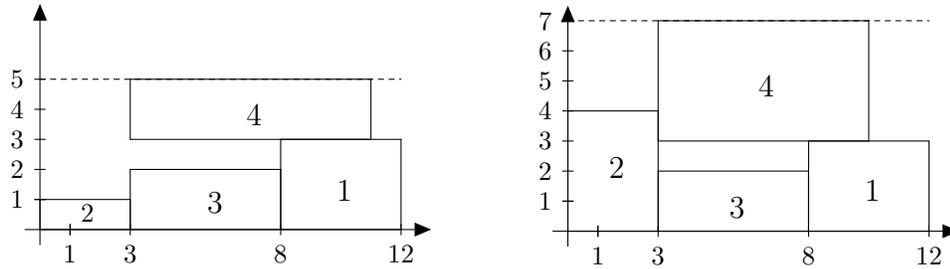


Figure 2.3: Example of a feasible non-preemptive schedule showing the processing of activities for two resources R_1 (left) and R_2 (right).

2.2 Maintenance Scheduling

Traditionally, machine maintenance was performed on a run-to-failure basis. In the late ‘50s and early ‘60s, with the recent developments in operational research, **preventive maintenance** (PM) was introduced and maintenance planning was formulated as any other ordinary scheduling optimisation problem. Tasks are to be performed (inspections and perhaps unexpected repairs), with a certain frequency and some capacity and workforce constraints. While these models do reduce system failure and dangerous accidents; they are labour intensive, do not account for the uncertainty of failure, and induce several unnecessary costs. The next step was towards **prognostic** models as they mitigate these issues by using the current “health”-state of the machine as an impetus factor to predict when failures are likely to occur and how to best avoid them. Conditioning the modelling on the current state of the system is where a relatively recent and active research area comes into play: **condition-based maintenance** (CBM).

In the airline framework, around “80% of the inspection and access activities do not lead to a repair”– Papakostas et al. (2010). This highlights the need to challenge and consider current maintenance scheduling models to reduce unnecessary inspec-

tions. Although some airlines are shifting to a more effective scheme through the use of sensors and more sophisticated analysis of the data, many still schedule maintenance periodically according to the opinion of their experts. This leads to suboptimal schedules vulnerable to the very likely introduction of delays and additional costs when demanding an unscheduled repair.

The techniques and approaches concerning the scheduling of different maintenance interventions, focussing mainly on the aircraft related, will be summarised in this section. As already mentioned in Section 1.1, there are considerable savings to be made from more efficient maintenance programmes. Furthermore, the improvements made can then be applied to other industries and areas that require such efficiency analysis like nuclear plants, automotive and other machinery operated in high-performance environments (e.g. hydraulic structures, brake linings, pipelines or cutting tools).

2.2.1 Prognostic-Based Models

Being able to replace optimally, at the most appropriate moment in time, the necessary components of the machine is essential to plan the maintenance workload and inventory reducing losses and general inefficiencies. By introducing the **remaining useful life** (RUL) of a system, prognostics can be defined to be the capability of predicting the changes in RUL. The RUL is, as its name suggests, the time until the system cannot perform its usual operations. Specifically, taking into account the current RUL of a system for every decision point is modelled under CBM.

Heng et al. (2009) and Jardine et al. (2006) presented very useful summaries on some of the current prognostic-based approaches and their corresponding benefits and

drawbacks. Particularly relevant to jet engines, in the former, the authors reviewed the methods for predicting rotating machinery failures. Besides, they both agreed that the most efficient and accurate models are CBM models. In this context, maintenance tasks are usually classified into three categories, proactive maintenance (inspections), PM or **corrective maintenance** (CM). CBM models recommend maintenance decisions based on the information collected through non-intrusive monitoring or updated historical data. CBM models also attempt to keep unnecessary operations (inspections and PM) to a minimum by performing an intervention exclusively when there is significant evidence of abnormal behaviour. A CBM program, thus, can help to reduce maintenance associated costs by reducing the number of unnecessary operations.

Jayabalan and Chaudhuri (1992) developed one of the first prognostic-based models. The authors introduced two types of maintenance interventions, which correspond to PM and CM. For each of these operations performed, a constant improvement factor γ , regarding the failure rate of the system is used. That is, every time a PM intervention is performed at a particular time t_i , the measure that quantifies the failure rate (e.g. the age of the system), is reduced by $t_i - t_i/\gamma$. As time increases, the cost of performing a corrective replacement also increases. Interventions of this type are assumed to restore the system to as-good-as-new state. Therefore, the authors employed a threshold such that if the failure rate reaches a certain level λ_{\max} , either a PM or CM operation will be performed. Based on the objective of minimising the total maintenance costs, one operation is chosen every time this occurs. Using this method and resting on several assumptions (intervention durations can be assumed to

be negligible), the authors formulated a cost model and presented a recursive equation to calculate the time for the n -th intervention t_n . It is given by the following equation,

$$t_n = t_{n-1} + \left(1 - \frac{1}{\gamma}\right)^{n-1} t_1 = \sum_{i=1}^n \left(1 - \frac{1}{\gamma}\right)^{i-1} t_1,$$

where t_1 is the time when the system reaches the maximum failure rate λ_{\max} , for the first time. For the case when $\gamma \geq 1$, depicted in Figure 2.4, the successive maintenance intervals are of decreasing length. The times between the interventions decreases as the system deteriorates faster after every intervention. At each of these points t_i , a cost problem is solved to determine whether a PM or a CM should be performed. In the case a CM is required to minimise the total cost, the failure rate will drop to 0 (not shown).

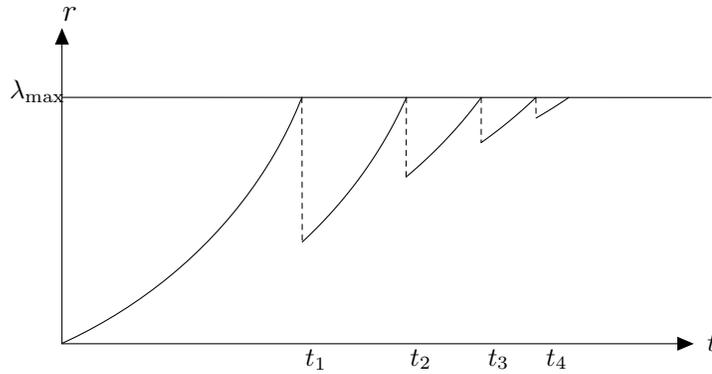


Figure 2.4: Failure rate (r) evolving through time (t) after performing PM interventions at times t_i for $i = 1, 2, 3, 4$ (assuming $\lambda \geq 1$).

Grall, Bérenguer and Dieulle (2002) introduced the use of N thresholds. These thresholds are used to generate aperiodic inspection schedules for maintenance in a similar fashion as above; however, the type of intervention performed depends on the system state. The system state is defined as a stochastic deterioration process follow-

ing a gamma distribution. Authors argued that the number of thresholds should be allowed to vary depending on how fast a system deteriorates; and suggested that if a system degrades quickly, the model benefits from having a large number of thresholds, and if a system deteriorates slowly, then fewer thresholds should be used. Sets of intervals for the thresholds are studied depending on the characteristics of the problem. These are then varied according to the objective function value and corresponding optimality criteria. The assumption that CM interventions restate the machine to a condition as-good-as-new is also accepted in other prognostic models, Grall, Dieulle, Bérenguer and Roussignol (2002) for instance. Marseguerra et al. (2002) highlighted the inefficiencies of this malpractice, including them being a financial burden, and presented a non-analytical model to deal with partial recovery using Monte Carlo simulations.

CBM-based Formulation for Fighter Aircraft Maintenance (Li et al., 2016)

One of the current CBM model formulations for fighter aircraft is the one given by Li et al. (2016). This formulation combined the “health”-state monitoring and several uncertainty measures in a single model. It is standard practice in the aircraft-related literature to take such “health”-state to be the legal remaining flying hours (Sarac et al., 2006). By updating the estimated remaining available flying hours for an individual aircraft, i.e. the RUL, according to certain historical information about the aircraft, the authors proposed a way to model the problem. The uncertainties included in this model account for the maintenance and flight durations drawn from uniform and normal distributions, respectively. The assumption that flight duration, mainte-

nance task durations, and even the departure delay come from normal distributions (with different parameters) is a conventional assumption in the literature. Based on the work by Mueller and Chatterji (2002) who, additionally, showed that departure delay can be modelled using a Poisson distribution.

Since Li et al. (2016) focused on scheduling maintenance for fighter aircraft, they use different types of flight operations, encapsulated by grouping the flights by “sorties”. This is due to various sorties or missions (e.g. transport, reconnaissance, bomber, fighter, or special operations) require different engine functions and manoeuvres.

To study the formulation in Li et al. (2016), the introduction of their notation is required. Let us define some sets and parameters,

\mathcal{F} : Set of all aircraft $i \in \mathcal{F}$, $|\mathcal{F}| = n$ represents the number of aircraft;

\mathcal{J} : Set of all sortie types $j \in \mathcal{J}$, $|\mathcal{J}|$ represents the number of unique sortie types;

l_j : Duration of sortie type $j \in \mathcal{J}$;

h_{\max} : Maximum number of flying hours between maintenance interventions;

h_j^t : Life of aircraft $i \in \mathcal{F}$ remaining at period t ;

k : Average number of time period required to complete maintenance;

b_i : Flying hours remaining on aircraft $i \in \mathcal{F}$ at the beginning of the time horizon;

M : Maximum number of aircraft that can be in maintenance at any given period t without penalty.

The decision binary variables are x_{ij}^t and m_i^t for the aircraft and maintenance respectively and are defined as follows,

x_{ij}^t : 1, if aircraft $i \in \mathcal{F}$ flies sorties type $j \in \mathcal{J}$ in period $t \in \mathcal{T}$; 0, otherwise;

m_i^t : 1 if aircraft $i \in \mathcal{F}$ enters maintenance in period $t \in \mathcal{T}$; 0, otherwise.

Time is taken to be discrete, with a time horizon denoted by T , thus $t \in \mathcal{T} = \{1, \dots, T\}$. The discrete-time unit for this model is half-days, that is, when t goes up by a single unit this corresponds to an increase in 12 hours. The problem can now be formulated as follows,

Model 2.2.1. *Time-discrete MIP with CBM considerations (Li et al., 2016)*

$$\min_m \max \sum_i \sum_{\tau=t-k} (m_i^\tau - M), \quad \forall t \in [k+1, T] \quad (2.2.1a)$$

Subject to

$$h_i^1 = b_i, \quad \forall i \in \mathcal{F} \quad (2.2.1b)$$

$$\sum_j x_{ij}^t l_j + h_i^{t+1} \leq h_i^t + h_{max} m_i^t, \quad \forall t \in \mathcal{T}, i \in \mathcal{F} \quad (2.2.1c)$$

$$\sum_j x_{ij}^t l_j + h_i^{t+1} \leq h_i^t, \quad \forall t \in \mathcal{T}, i \in \mathcal{F} \quad (2.2.1d)$$

$$h_{max} m_i^t \leq h_i^{t+1} \leq h_{max}, \quad \forall t \in \mathcal{T}, i \in \mathcal{F} \quad (2.2.1e)$$

$$h_{max} m_i^t \leq h_{max} - h_i^t, \quad \forall t \in \mathcal{T}, i \in \mathcal{F} \quad (2.2.1f)$$

$$\sum_j x_{ij}^t \leq 1, \quad \forall t \in \mathcal{T}, i \in \mathcal{F} \quad (2.2.1g)$$

$$\sum_i x_{ij}^t \leq 1, \quad \forall t \in \mathcal{T}, j \in \mathcal{J} \quad (2.2.1h)$$

$$\sum_j x_{ij}^{t+y} + m_i^t \leq 1, \quad \forall t \in [1, T-k+1], y \in [0, k-1], i \in \mathcal{F} \quad (2.2.1i)$$

$$m_i^t + m_i^{t+y} \leq 1, \quad \forall t \in [1, T-k], y \in [1, k], i \in \mathcal{F}, t \in \mathcal{T} \quad (2.2.1j)$$

$$h_i^t \geq 0; x_{ij}^t, m_i^t \in \{0, 1\}, \quad \forall t \in \mathcal{T}, i \in \mathcal{F} \quad (2.2.1k)$$

The objective function 2.2.1a, minimises the maximum number of aircraft in service during any average maintenance duration period and ensures this does not exceed

the maintenance capacity M . Constraints 2.2.1b initialise the remaining flying hours for each aircraft. This initialisation is critical, otherwise, the formulation leads to a contradiction. To include stochasticity, the model draws each initial number of flying hours b_i , from a uniform distribution with certain parameters. The authors draw from a $U(0, 200)$. Constraints 2.2.1c, 2.2.1d, and 2.2.1e update the remaining flying hours for aircraft i for the next time period $t + 1$, depending on sorties flown and whether maintenance is currently being performed or has been performed. Constraints 2.2.1f guarantee that maintenance has to be performed when the remaining flying hours for an aircraft reaches the fixed level h_{\max} . Constraints 2.2.1g enforce that at most one sortie per aircraft. Constraints 2.2.1h ensure that no sortie is assigned to more than one aircraft. Both constraints 2.2.1i and 2.2.1j force the downtime of an aircraft to be sufficient to allow the completion of maintenance, this involves the maintenance duration uncertainty captured by k . That is, if maintenance is to be performed for aircraft i at time period t , so $m_i^t = 1$, then maintenance cannot be performed again until after a period greater than the average duration of said maintenance operation, $m_i^{t+y} = 0$ for $y \in [1, k]$. Similarly, if $m_i^t = 1$, then aircraft i cannot be allocated to operate any sortie j until after a period greater than the average duration of a maintenance task and therefore $\sum_j x_{ij}^{t+y} = 0$. The authors assumed that maintenance duration k , follows a $\mathcal{N}(3, 1)$. Lastly, constraints 2.2.1k define the domains of the decision variables.

In the case study presented, with time horizon $T = 50$ (or 25 days), $n = 200$ aircraft, two types of sorties ($|\mathcal{J}| = 2$), and $M = 0$, the model minimises the maximum number of aircraft undergoing maintenance at any given period. Although the com-

putational times were quite promising, the model presented several limitations. The remaining flying hours were disregarded for the end of the time horizon. Nonetheless, one would not expect most of the aircraft to become non-operational for the next planning period. Moreover, the “health”-state or condition of the aircraft was included by constraints which do not account for the possible relationship between the different factors in operations history. Finally, letting $M = 0$ with the objective function as formulated, leads to aircraft with low remaining flying hours being held at the base without operating any sorties.

Alternative Generation for Aircraft Maintenance Planning (Papakostas et al., 2010)

A different modelling approach, which removes the issues previously discussed for the formulation by Li et al. (2016), was given by Papakostas et al. (2010). By combining the decision-making and alternative generation approach from Chryssolouris et al. (1992), and the set-partitioning based IP formulation from Sarac et al. (2006); the authors presented a CBM methodology for the scheduling of aircraft maintenance tasks at an operational level or short-term planning horizon.

Chryssolouris et al. (1992) introduced a generic decision-making framework for the allocation of different resources, in a manufacturing system, to production tasks. Such framework involved the application of a simulation approach at every “decision point”. Here, decision points are defined as points in time when tasks are completed. Notably, they develop an algorithm for the generation of alternatives for every decision point.

In the airline operator setting, as proposed by Papakostas et al. (2010), the decision-making steps at every decision point are as follows,

Step 1. Identify required maintenance tasks.

Step 2. Determine decision criteria and weights for evaluating alternative.

Step 3. Generate alternatives.

Step 4. Determine the consequences of the different alternatives and their utility.

In this case, a decision point is defined as a point in time when an aircraft has landed and there are several components to be scheduled for maintenance. In step 1, for the identification of the required maintenance tasks involved with a certain alternative, a set of constraints for maintenance and time requirements have to be imposed and updated to form the decision-making criteria in step 2. Such constraints include, a number of suitable airports for each task, manpower available to perform each task, and flight arrival and departure times. The authors borrowed parts of the formulation by Sarac et al. (2006) to achieve this. In step 3, alternatives are defined as allocations of maintenance interventions to suitable resources either at the current or successive workshops. For the alternative generation, it is important to establish a maximum number of alternatives (MNA) to be considered and a sampling rate (the number of times a single alternative is considered). The methodology proposed generates MNA alternatives for every decision point (using the algorithm by Chryssolouris et al., 1992), and each one of these alternatives is simulated SR times.

To present a schedule from these feasible alternatives, the authors introduced an

utility measure for every alternative in step 4. Several factors influence such utility measure. These include the cost, operational risk, flight delay and RUL; each of which has an individual weight (step 2). Once this has been calculated for the MNA alternatives (each simulated SR times), the tasks can be ordered by the utility. Thus, it can effectively be seen which tasks should be performed first. Each of these factors can be calculated as outlined below.

Cost criterion. The cost criterion depends on several independent costs for each workshop. The cost of performing task T_y (out of the total number of tasks Y) at workshop i of airport j , is summarised in the following formula

$$\text{Cost} = (\text{E} + \text{L} + \text{OH}) \times \text{O} + \text{C}$$

where E = equipment rate, L = labour rate, OH = overhead rate, O = time required for the completion of a task, and C = procurement cost associated with the same task (parts and additional materials). All of these are dependant on the type of task T_y and on the workshop i of airport j , denoted with R_{ij} .

The cost of an alternative k , Al_k , is given by

$$\text{Cost}(Al_k) = c_{1,k} = \frac{\sum_{s=1}^{SR} \sum_{y=1}^Y \text{Cost}(T_y, R_{ij})_s}{SR}$$

where the cost function $\text{Cost}(T_y, R_{ij})_s$, dependent on task type and airport workshop, is evaluated for each s -th sample of the alternative.

Operational risk criterion. The operational risk accounts for the stochasticity

of events: desirable and undesirable. Desirable events are taken as the scheduled events. Unwanted or undesirable events include additional hiring or equipment failure and can be regarded as considering the worst case scenario. The overall uncertainty can be included by taking the expected value of the gain of a certain maintenance task allocation. The formula, therefore, sums the desirable costs (as calculated from the cost criterion) times its corresponding success probability, plus the undesirable costs times the probability of these events happening. The formula is given by,

$$\text{Risk}(T_y, R_{ij}) = \text{Cost}(T_y, R_{ij}) \cdot \text{SP}(T_y, R_{ij}) + \text{UnDV}(T_y, R_{ij}) \cdot \text{UnDP}(T_y, R_{ij}) .$$

The cost of an unscheduled or undesirable event UnDV is taken to be twice as much as the desirable scheduled event cost. Both the success probability (SP) and undesirable probability (UnDP) can be taken from two separate Weibull distributions according to the updated failure state of the aircraft. Using Weibull analysis is common practice when estimating lifetimes and failures for different types of machinery.

The total operational risk for an alternative k is the average over all the samples, hence,

$$\text{Risk}(Al_k) = c_{2,k} = \frac{\sum_{s=1}^{SR} \sum_{y=1}^Y \text{Risk}(T_y, R_{ij})_s}{SR} .$$

Flight delay criterion. The possible future flight delay depends on three independent stochastic events. The duration of flights, the duration of a maintenance task and departure delay. With this in mind, the authors draw the probabilities for each of these events from different normal distributed random variables with different mean

and variance. Delay in a certain flight leg l will be introduced by unsuccessful delivery of a maintenance intervention (usually because of lack of resource availability) or by weather conditions or extreme events. Delays will then be propagated throughout the day, influencing future departure and arrival times.

The flight delay for a certain alternative is given by

$$\text{FD}(Al_k) = c_{3,k} = \frac{\sum_{s=1}^{SR} \sum_{y=1}^Y \{\max[AT_{l,s} - AT_l^{\text{due}}, 0] + \max[DT_{l,s} - DT_l^{\text{due}}, 0]\}}{SR}$$

where $AT_{l,s} - AT_l^{\text{due}}$ is the difference between the actual arrival time and the planned arrival time and the second term is the equivalent for departure times. Note the similarity between this criterion and the earliness/tardiness cost, objective function (iii), discussed in Section 2.1.2.

RUL criterion. The remaining useful life criterion simply corresponds to a health assessment prediction for an aircraft quantified by the remaining flying hours. This figure can be calculated from a stochastic model that takes into account the interdependence of the components which are monitored through different sensors. This quantity is taken as initially known and consequently updated. By considering the change in RUL when different tasks are performed, the RUL for an alternative k can be calculated as,

$$\text{RUL}(Al_k) = c_{4,k} = \frac{\sum_{s=1}^{SR} \sum_{y=1}^Y wr_y |\text{RUL}(T_y, R_{ij})_s|}{SR},$$

where $\text{RUL}(T_y, R_{ij})$ is the remaining useful life of task T_y when it is allocated to

resource R_{ij} . A weight wr_y , is associated to each task y , such that the relative importance (with respect to the other tasks) of performing a certain task is captured. These weights can be reassigned at step 2 after a historical data set has been gathered and examined.

Utility measure. Once all the criteria have been calculated, they can be normalised and according to a fixed set of criteria weights, w_c , attributed to the utility gain for each criterion ($c = 1, 2, 3, 4$), the total utility can be quantified. These criteria weights, assigned in step 2 of the decision-making process, can be initialised according to some given priorities and then be updated according to how the system responds. For the case study in the paper, these were taken as 30% for the cost, 20% for the operational risk, 10% for the flight delay and 40% for the RUL ($w_1 = 0.3$, $w_2 = 0.2$, $w_3 = 0.1$ and $w_4 = 0.4$). The utility for alternative k is given by

$$\text{Ut}(Al_k) = \sum_{c=1}^4 w_c \frac{c_c^{\max} - c_{c,k}}{c_c^{\max} - c_c^{\min}}.$$

Where, c_c^{\max} and c_c^{\min} correspond to the maximum and minimum values of criterion c over all the alternatives. This provides the normalisation,

$$\text{Ut}(Al_k) \in [0, 1] \forall k \in \{1, \dots, \text{MNA}\}.$$

The alternative with the best utility will, ultimately, provide the best possible allocation of all tasks at current or successive airport workshops for all aircraft under examination. Given how the utility function is defined, a greater utility implies the

weighted minimisation over all criteria.

The computational analysis for the case study included a MNA of 49 alternatives and a sampling rate, a simulation rate (SR) of 10, and provided a solution within 20 minutes. If the alternative with the highest utility is chosen, the solution is guaranteed to provide an optimal, feasible and robust schedule. The repeated sampling from the different distributions for each of the sampling stages, provides a robust solution.

2.2.2 Modelling Scheduling Problems in Other Industries

This section outlines some of the strategies several authors have adopted for other problems in the literature. Initially, problems where resource allocation plays a crucial role are reviewed, specifically, applied to cloud computing and virtualised data centres. As briefly mentioned, prognostic models can also be applied in the context of electronics, production plants, gas turbines, cars, and general systems subject to vibrations.

Urgaonkar et al. (2010) addressed the power management issues for the so-called *virtualised data centres* by employing Lyapunov optimisation to ensure optimal allocation of resources. The dynamic resource allocation is an appropriate way of modelling for this problem, as at any point in time there are a different number of applications that compete for the available processing servers. Furthermore, these applications are heterogeneous, that is, they have individual processing requirements (time and memory). In the context seen for the RCPSP, what was referred to as jobs or requests, can now be taken as the processing of applications, and the resources are servers. The aim of the authors was to develop an algorithm that provided a robust resource allocation

solution while coping with time-varying workloads. The algorithm exploited queuing data from the arrivals of processing requests to infer about future arrival processes. The arrival of processes were taken from a random stochastic process with a certain time average rate; however, no assumption was made for the distribution of these. The authors employed queuing dynamics and backlog information to estimate the arrival process. For the inclusion of the power management element, the model was designed to adapt to electricity consumption variations from different CPU speeds and usages. The power-frequency relationship that derivates from dynamic voltage and frequency scaling for the CPU's is non-linear. By using Lyapunov optimisation, the authors not only dealt with the complexity and non-linearity of the problem effectively, but also derived analytical performance guarantees of the algorithm.

Christer et al. (1998) employed a statistical approach to model the prediction of faults for an *extrusion press*. Assuming some distributional properties for the faults, the authors implemented a planned PM approach. The aim was to reduce the costs and interruptions to the production process. The fault origination process was modelled by a homogeneous Poisson process. This makes expected number of failures easy to calculate. To make the model more realistic, the authors used available historical data to make the expected number of failures increase over time as the extrusion press deteriorates. The delay time concept plays a crucial role in the modelling of PM planning in this model. It is defined as the time between when a fault is detected and the time that the machine fails. Combining the updated proportion of failures (after some inference from the data), an explicit distribution function for the delay time was given. More specifically, the cumulative distribution function for the delay

time was shown to be a weighted sum of uniform and exponential distributions.

Some authors have applied artificial intelligence techniques for finding prognostic predictions. Neural networks for example, have been implemented on prognostic predictions for gas turbines by DePold and Gass (1998) and for cars by Jhwei et al. (2008). A brief discussion on the use of artificial neural networks for airline crew scheduling is given further on, in Section 2.3.2. Also, wavelet theory has been employed for condition monitoring and fault diagnostics (Peng and Chu, 2004).

2.3 Airline Scheduling and Recovery

The airline scheduling problem consists of several steps with different planning horizons corresponding to airlines' planning stages. These are, flight scheduling, fleet assignment, tail assignment, and, crew scheduling. These stages are traditionally modelled is by solving them sequentially, the output of one stage is input for the next. This scheme is summarised in Figure 2.5. Firstly, a flight schedule needs to be provided, usually 12 months in advance, according to demand and other restrictions an airline wishes to apply. Different fleet types are then assigned to certain routes or flight legs in a way that maximises the revenue, and that does not violate safety or maintenance regulations. This stage is referred to as the fleet assignment problem, and is usually solved for a 12-week horizon. Given the allocation of fleet types to flights, the next stage determines a 1-4 week sequence of flights to be flown by each aircraft. The solution must ensure that each flight is flown exactly once, each aircraft is maintained as per regulations, and capacity constraints for fleet types are satisfied

(Cadarso et al., 2016). Some authors refer to this stage as the **tail assignment** (TA) problem or aircraft routing/rotation. At the final stage, crew scheduling uses the solution from the fleet assignment to provide a 1-4 week schedule for the airline crew; allocating flight legs, working hours, and satisfying duty regulations. Typically, this is done in two stages, crew *pairing* and crew *rostering*. The former assigns the required flight legs, or pairings, to be operated by some appropriate crew members. While the latter, creates rosters for each individual crew member, with additional working hours and duty regulations (Kohl and Karisch, 2004).

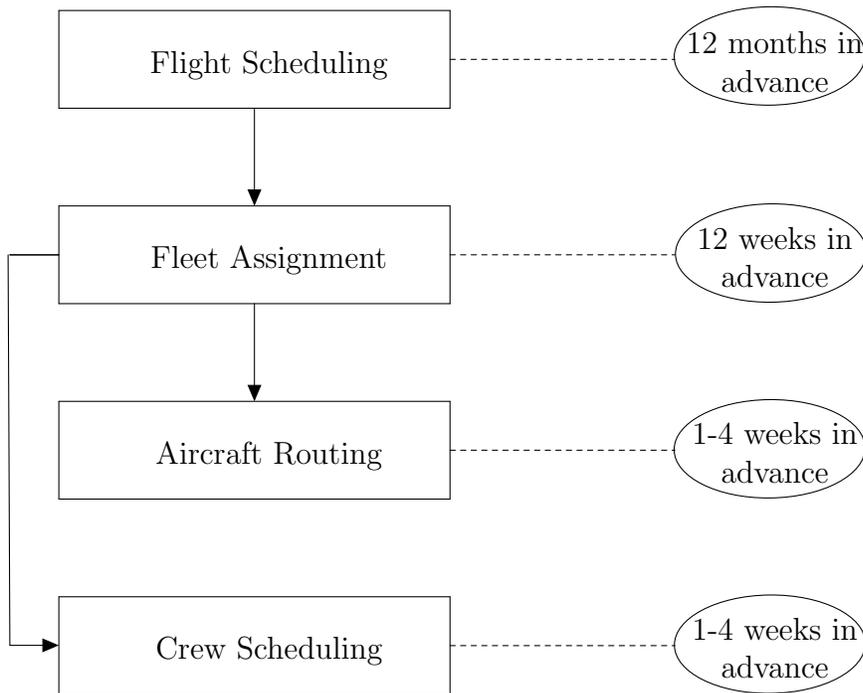


Figure 2.5: Sequential modelling of the airline scheduling problem (Bae, 2010).

In contrast, the airline recovery problem considers the same steps but in an operational setting. In this scenario it is assumed that pre-operational plans have been unpredictably disrupted, therefore, the aim is to propose alternative plans that al-

low a fast resumption of airline operations while reducing the impact caused by the deviation from the original pre-operational schedule. Such impact can be quantified by introduced passenger delay, or additional recovery costs (e.g. crew costs and regulation fines). Given the nature of the problem, the time horizon for these problems is significantly reduced to an operational horizon, between 1-10 days. The steps in airline recovery are, flight recovery (sometimes schedule recovery), fleet recovery, **aircraft recovery** (AR), crew recovery, and, additionally, passenger recovery. The latter, not present in the airline planning stages, consists on the rerouting of passengers, by using different means of transportation if necessary, to deliver them to their destination. The characteristics of the remaining problems is the same as their pre-operational counterpart, but with disruption considerations. Moreover, the order, when solving them sequentially, is also the same.

In the coming sections, we describe, in detail, the literature for some components of the airline scheduling and recovery problems. Specifically, in Section 2.3.1, we focus on the TA and AR problems, and, in Section 2.3.2, we provide a review of the integrated airline scheduling and recovery literature.

2.3.1 Tail Assignment and Aircraft Recovery

TA, sometimes also referred to as aircraft rotation, is a combinatorial optimisation problem that deals with the assignment of individual aircraft to different flight legs. AR, considers the same problem but with flight cancellation and delay considerations. There are three main types of formulation for these problems, **string-based** models, **multi-commodity network flow** (MCNF) models, and **time-space network**

(TSN) models.

String-based models are a type of formulation that formulates the TA problem using strings, i.e. sequences of connected flights that begin and end at a maintenance workshop, and, that satisfy flow balance and maintenance regulations (Barnhart, Boland, Clarke, Johnson, Nemhauser and Shenoi, 1998).

MCNF models, introduced by Feo and Bard (1989) for the TA problem, and first used by Argüello (1997) for the AR problem, are based on a fleet-flow time-space network (layered TSN models). Here, each aircraft represents separate commodities and flow has to be preserved. The formulation, typically, includes capacity constraints (passengers and fleet) and conservation of aircraft, flight, and airport flow.

TSN models were introduced by Jarrah et al. (1993) for AR problem and then used by Clarke et al. (1997) for the TA problem. As its name suggests, in a TSN, each airport is represented by a time line where nodes show every departure/arrival at the corresponding point in the airport time line, and arcs show flights and connections.

For convenience and brevity, let us study each of the three different formulations for the TA problem.

String-based Formulation (Barnhart, Boland, Clarke, Johnson, Nemhauser and Shenoi, 1998)

The first string-based formulation was presented by Barnhart, Boland, Clarke, Johnson, Nemhauser and Shenoi (1998) for the TA problem. To study this formulation, let us introduce their notation,

F : Set of flights indexed by i ;

K : Set of fleet indexed by k ;

G_k : Set of ground connections for fleet k indexed by j ;

S : Set of strings indexed by s ;

S_i^- : Set of augmented strings ending with flight i ;

S_i^+ : Set of augmented strings starting with flight i ;

a_{is} : Parameter which is equal to 1 if flight i is contained in string s , 0 otherwise;

c_s^k : Parameter representing the cost of assigning string s to fleet k .

Two variables are necessary, x_s^k is a binary variable which is equal to 1 if string s is flown by fleet k ; or, 0 otherwise. The continuous variable, $y_{(\cdot)}^k$ is equal to the number of aircraft of fleet k on the ground at the station between certain events in the subscript.

For instance, the number of aircraft of fleet k on the ground at the station between the predecessor event of i and the arrival (departure) of i are denoted by $y_{(e_{i,a}^{-,k}, e_{i,a}^k)}^k$ $\left(y_{(e_{i,d}^{-,k}, e_{i,d}^k)}^k \right)$; and, between the arrival (departure) of i and its successor, by $y_{(e_{i,a}^k, e_{i,a}^{+,k})}^k$ $\left(y_{(e_{i,d}^k, e_{i,d}^{+,k})}^k \right)$.

Model 2.3.1. *String-based formulation for the TA problem (Barnhart, Boland, Clarke,*

Johnson, Nemhauser and Shenoi, 1998).

$$\min \sum_k \sum_s c_s^k x_s^k \quad (2.3.1a)$$

Subject to

$$\sum_k \sum_s a_{is} x_s^k = 1 \quad \forall i \quad (2.3.1b)$$

$$\sum_{s \in S_i^+} x_s^k - y_{(e_{i,d}^-, e_{i,d}^k)}^k + y_{(e_{i,d}^k, e_{i,d}^+, k)}^k = 0 \quad \forall i, k \quad (2.3.1c)$$

$$- \sum_{s \in S_i^-} x_s^k - y_{(e_{i,a}^-, e_{i,a}^k)}^k + y_{(e_{i,a}^k, e_{i,a}^+, k)}^k = 0 \quad \forall i, k \quad (2.3.1d)$$

$$\sum_s r_s^k x_s^k + \sum_{j \in G^k} r_j^k y_j^k \leq N_k \quad \forall k \quad (2.3.1e)$$

$$y_j^k \geq 0 \quad \forall k, j \in G_k \quad (2.3.1f)$$

$$x_s^k \in \{0, 1\} \quad \forall s, k \quad (2.3.1g)$$

The objective function 2.3.1a minimises the cost of the strings assigned. Constraints 2.3.1b ensure that each flight is in exactly one string. Constraints 2.3.1c and 2.3.1d assure that the flow of aircraft of arriving and departing from a certain location is conserved. Constraints 2.3.1e enforce that the total number of fleet k , the sum of the aircraft flying (first term) plus the ones on the ground (second term), does not exceed the size of the fleet k , N_k . This employs two additional parameters, r_s^k and p_j^k , which represent the number of aircraft in fleet k flying and on the ground, respectively, at a certain point in time. The point in time when these parameters are measured, authors argued, does not make a difference due to the flow constraints. Lastly, constraints 2.3.1f and 2.3.1g define the variables.

Some influential works employ the string-based formulation for the TA problem, Cohn and Barnhart (2003); Sarac et al. (2006); Papadakos (2009), among others. Mather (2015) has used this type of formulation to solve the integrated airline recovery

problem (discussed in more detail in Section 2.3.2). However, string-based formulations have a major disadvantage: generating all possible strings becomes intractable even for small instances. The reason for this is because they are non-compact formulations, that is, they do not have a polynomial number of variables and constraints. Therefore, these types of formulations are solved using a column generation framework. For more details on this method see Section 2.4.1.

Integer Multi-Commodity Network Flow Formulation (Feo and Bard, 1989)

The integer multi-commodity network flow formulation was proposed for the TA problem by Feo and Bard (1989). In order to study it in more detail, let us introduce their notation,

D : Set of days with $D = \{1, \dots, n_D\}$, indexed by d ;

A : Set of airports with $A = \{1, \dots, n_A\}$, indexed by j or k ;

A_d : Set of airports on the evening of day d , where $A_d \subseteq A$;

K : Set of aircraft with $K = \{1, \dots, n_K\}$, indexed by i ;

c_j : Unit cost for performing maintenance at airport j ;

p_j : Maintenance capacity for airport j (in number of aircraft).

The variable x_{ijk} is equal to 1 if aircraft i is in city $j \in A_d$, and in city $k \in A_{d+1}$; 0 otherwise. The variable $w_{ij(d)}$ is equal to 1 if aircraft i receives maintenance at city $j \in A_d$. The integer MCNF formulation is as follows,

Model 2.3.2. *Integer MCNF formulation for the TA problem (simplified) (Feo and Bard, 1989).*

$$\min \sum_{i=1}^{n_K} \sum_{d=1}^{n_D} \sum_{j \in A_d} c_j w_{ij} \quad (2.3.2a)$$

Subject to

$$\sum_{j \in A_{d-1}} x_{ijk} - \sum_{j \in A_{d+1}} x_{ikj} = 0 \quad \forall i, k \in A_d \quad (2.3.2b)$$

$$\sum_{i=1}^{n_K} x_{ijk} = 1 \quad \forall d, (j, k) \in A_d \times A_{d+1} \quad (2.3.2c)$$

$$\sum_{i=1}^{n_p} w_{ij} \leq p_j \quad \forall d, j \in A_d \quad (2.3.2d)$$

$$w_{ij} - \sum_{k \in A_{d-1}} x_{ikj} \leq 0 \quad \forall i, j \in A_d \quad (2.3.2e)$$

$$x_{ijk} \in \{0, 1\} \quad \forall i, j, k \quad (2.3.2f)$$

$$w_{ij} \in \{0, 1\} \quad \forall i, j \quad (2.3.2g)$$

The objective function 2.3.2a minimises the total maintenance cost. Constraints 2.3.2b ensure the conservation of flow for each aircraft (commodity) and airport-day pair. Constraints 2.3.2c ensure that the number of aircraft assigned to a flight is exactly one. Constraints 2.3.2d restrict the number of aircraft that can have maintenance at an airport i on day d . Constraints 2.3.2e represent the conservation of flow for maintenance. Constraints 2.3.2f define the domains of the variable. Authors included some additional constraints for maintenance which we have omitted to simplify the formulation.

The integer MCNF formulation has been chosen by multiple authors to solve the TA and AR problems (Bard et al., 2001; Yan and Tseng, 2002; Sriram and Haghani, 2003; Mercier et al., 2005; Eggenberg et al., 2010). Even though this formulation is compact, having a polynomial number of variables, the number of variables is

$O(n_A^2 n_K)$ which is quite limiting for large instances.

Time-Space Network Formulation (Clarke et al., 1997)

The time-space network formulation was offered for the TA problem by Clarke et al. (1997). To discuss the formulation in more detail, let $G = (N, A)$ be a graph with a set of nodes N and arcs A . For each arc $i \in A$ let us denote the head and tail functions as $h(i)$ and $t(i)$, respectively. Let v_{ij} denote the through value of flight j following i ; if, either i or j is a ground arc, then it takes the value 0. We require a binary variable, x_{ij} , which is equal to 1 if flight i is connected to flight j , and 0 otherwise.

Model 2.3.3. *TSN formulation for the TA problem (Clarke et al., 1997).*

$$\max \sum_{\{i,j \in A: h(i)=t(j), i \neq j\}} v_{ij} x_{ij} \quad (2.3.3a)$$

Subject to

$$\sum_{\{j: h(j)=t(i), i \neq j\}} x_{ij} = 1 \quad \forall i \in A \quad (2.3.3b)$$

$$\sum_{\{i: h(i)=t(j), i \neq j\}} x_{ij} = 1 \quad \forall j \in A \quad (2.3.3c)$$

$$\sum_{\{i \in S, j \in A \setminus S: h(i)=t(j)\}} x_{ij} \geq 1 \quad \forall S \subset A \text{ with } 2 \leq |S| \leq |A| - 2 \quad (2.3.3d)$$

$$\sum_{\{i \in P', j \in A \setminus f(i): h(i)=t(j)\}} x_{ij} \geq 1 \quad \forall P \subset P^k, k \in K \quad (2.3.3e)$$

$$x_{ij} \in \{0, 1\} \quad \forall (i, j) \in A \times A \quad (2.3.3f)$$

The objective function 2.3.3a, maximises the through value for all connecting flights. Constraints 2.3.3b ensure that exactly one connection is chosen between each flight i and its corresponding neighbours j , i.e. $h(j) = t(i)$. Conversely, constraints 2.3.3c ensure that for every flight j there is exactly one connection to its corresponding

neighbours i , i.e. $h(i) = t(j)$. Constraints 2.3.3d (subtour elimination constraints) ensure that no cycles are present in the solution. Constraints 2.3.3e ensure that no path, P , out the set of paths with maintenance violations for maintenance of type k , P_k , is allowed. Constraints 2.3.3f define the domains of the variables.

TSN models have been widely applied to the TA and AR problems (Thengvall et al., 2000, 2001; Løve et al., 2001; Hicks et al., 2005; Bratu and Barnhart, 2006; Orhan et al., 2012; Haouari et al., 2013; Liang and Chaovaitwongse, 2013; Safaei and Jardine, 2018; Khaled et al., 2018). The shortcoming of this TSN formulation is that individual aircraft are not considered. Hence, when maintenance is considered, aggregated maintenance constraints have to be implemented. Khaled et al. (2018) removed this restriction by introducing an index for every aircraft. Additionally, the authors provided a compact formulation with $O(|A||K|)$ variables; as opposed to $O(|A|^2)$ variables in the original formulation. The additional aircraft index, allows for more accurate maintenance modelling, specifically, maintenance checks of type A are accounted for. Nevertheless, some restrictive assumptions are placed on maintenance activities. For instance, that maintenance can only be performed at night.

2.3.2 Integrated and Semi-Integrated Airline Scheduling and Recovery

The airline scheduling problem involves solving an array of problems that facilitate airlines' daily operation. Similarly, the airline recovery problem solves a closely related sequence of problems that consider the mitigation of the effects caused by an

unforeseen disruption. As discussed in section 2.3, the common problems are, flight scheduling/recovery, fleet assignment/recovery, tail assignment or aircraft recovery, and crew scheduling/recovery. Airline recovery additionally considers passenger recovery. Clearly, when solved sequentially, the solution to one problem does not take into consideration the restrictions of the subsequent problems. Even though this sequential approach reduces the computational complexity, due to the interdependence of each stage, the solution provided is suboptimal, often far from optimality. Models have been developed to breach this gap and provide better quality results, which consider the combination of two or more stages into a single problem. Despite initially being considered computationally intractable (Hane et al., 1993), many fruitful and tractable models have emerged. Models are grouped under the labels **integrated** or **semi-integrated**, depending on whether all stages are integrated or not.

Integrated Airline Scheduling

The first semi-integrated model, integrating fleet assignment and aircraft routing, was presented by Desaulniers et al. (1997). In this paper, assuming the flight origin/destination information is given, authors introduced the use of time-windows on flight departures for the fleet assignment problem. The problem is modelled using two equivalent formulations: a set partitioning type formulation (Model 1) and a time constrained MCMF formulation (Model 2). Distinct solution approaches were used for the two different formulations. Column generation, benefiting from the equivalence of the formulations, is used to solve both the first and second models to achieve lower and upper bounds respectively. With these bounds the optimal solution is ob-

tained via branch-and-bound on Model 2. Apart from providing a solution within a reasonable time, for the largest number of aircraft considered the total CPU time was at most 1 hour, these solutions provided a significant improvement in the profit.

The semi-integrated model by Yan and Tseng (2002) integrated flight scheduling, fleet assignment and aircraft routing. Similarly, as the previous authors, they used an integer MCNF formulation. In this model, all the flight legs are considered as optional legs and are selected as required. An algorithm is developed to solve the problem. It is based on Lagrangian relaxation, the network simplex method, the least cost flow augmenting algorithm, and the flow decomposition algorithm. Given the computer processors used at the time of the study, the solution times for this algorithm are very high rounding 160 hours for moderately sized problems.

Sriram and Haghani (2003) presented a semi-integrated model for fleet assignment and aircraft routing. The optimisation formulation presented, extended from the pioneering work by Feo and Bard (1989), included heterogeneity of the fleet and two types of maintenance checks. The result was an integer MCNF formulation which accounted for maintenance checks A and B. The objective of the model is the minimisation of the costs associated with maintenance and invalid fleet assignment. The heuristic solution procedure provided, constructs schedules using a search technique based on depth-search first and random search. Establishing a maximum number of iterations terminates the heuristic solution procedure and provides a near-optimal solution. This solution method provided good quality solutions for a problem with 58 flights and 75 airports in 5 minutes of computational time.

Yan et al. (2004) addressed the issues that arise when general airline crew schedul-

ing models are applied to maintenance crew. The authors presented a MIP formulation that solved the short-term maintenance manpower scheduling problem. The objective of the model is to minimise the total manpower required. The constraints ensure that the demand for maintenance intervention types for each type of aircraft is met at every time period, taking into account the certificates that technicians possess. The certification system increases the complexity of the problem. Furthermore, authors included several models to allow for different flexible strategies, these include shifts, squad size, and working hours. The solution method developed includes splitting the initial problem into two subproblems, which are dealt with individually using a custom algorithm that efficiently selects “certificate combinations”. Given an initial feasible schedule, the algorithm (which is based on implicate enumeration) generates different certificate combinations which are selected by the minimum man-hour requirement. The procedure is repeated until a certain level of precision has been met. The algorithm was tested on a data set for 6 days, 6 types of aircraft (51 in total) with 481 maintenance items. The most complex model, taking all the flexible strategies criteria, provided a solution only 5% off optimality in under 11 minutes CPU time.

For the combination of aircraft routing and crew scheduling Mercier and Soumis (2005) provided a compact semi-integrated MIP formulation. The primary focus of this MIP is in crew scheduling, under the assumption that the fleet assignment problem has been solved. The three-phase algorithm presented to compute solutions, combined Benders’ decomposition method with a dynamic constraint generation procedure. The optimal solutions were shown to allow some degree of flexibility on the departure times. This induced considerable cost reductions and with no flight re-

timing, the CPU time for a single processor, on 510 flight legs, is just over 8 minutes. The effect of flight re-timing is also considered in the study and significantly increased the solution times to up to 36 hours. Flight re-timing is a way of balancing the fact that flight schedules are used as an input and allowed for flights to be rescheduled within certain time limits (for instance, ± 5 minutes).

A model that puts more emphasis on maintenance scheduling is given by Papadacos (2009). The formulation, based on the string-based MIP by Cohn and Barnhart (2003) integrated fleet assignment, aircraft routing, and crew scheduling. Two solution approaches were proposed one that just used Benders' decomposition and a novel "three-phase algorithm". The latter used a Benders' decomposition method (accelerated by the Magnanti-Wong method) combined with accelerated column generation. A case study examined data sets from an European airline (372 legs per day) and a North American airline (over 2100 legs per day), both composed of 6 types of aircraft and either **central hub** or **hub-and-spoke network** structure. These simply relate to the different connection types that airlines may have, for instance, a small airline is likely to have a central hub type network structure if the aircraft can only be serviced in one workshop. Low-cost airlines, on the other hand, typically have a hub-and-spoke network. Using these data sets, authors compared their two algorithms to the best-known method in the semi-integrated literature, which is given by Mercier et al. (2005). Out of the two heuristic algorithms, the more computationally expensive one, the three-phase algorithm provided the best solutions. However, both algorithms outperformed the best-known method. The largest instance considered, required a CPU time of 16.5 hours and did not reach optimality.

Cacchiani and Salazar-González (2016) concentrated on providing an exact solution for the problem of a particular real-world regional carrier. The MIP formulation proposed integrated fleet assignment, aircraft routing, and crew scheduling. This formulation takes maintenance into account but in a simple single machine scheduling type framework with some restrictive assumptions. Conditioned by the case study, the assumptions are that maintenance is only performed: at night, every 3 days, and in a single workshop. Similarly, as in the previous model, the authors developed a three-phase solution algorithm. Phase I obtains a lower bound on the solution by implementing column generation. Phase II derives an upper bound by heuristic methods (either a combination of Lagrangian relaxation and column generation or Benders' decomposition). Finally, Phase III uses a branch-and-price algorithm, similar to the one by Sarac et al. (2006), to achieve optimality. To speed up the algorithm, the authors employed an efficient way of completing the first stage of the algorithm with a new type of bounding cut. The case study visited was for the regional carrier provider Binter Canarias. Flying around the Canary Islands and as far as Marrakech-Menara, with up to 172 flights, the aircraft used are powered by turboprop engines. Although the meteorological conditions they are subjected to are sometimes extreme (the Canary Islands are known for their strong winds) the flights are short, between 30 and 70 minutes. The solution times for the largest case took approximately 2 hours. Moreover, the authors compared their algorithm with the heuristic algorithm currently in use by the airline company. The heuristic algorithm (previously developed by one of the authors, Salazar-González, 2014), was found to provide faster computational times in most instances; however, the exact method was found to improve the optimality

of the solution significantly. Regarding the objective function, between a 4.37% and 23.56% improvement were observed.

Cadarso et al. (2016) addressed the influence of competition to flight schedules, airline revenues, and destinations. It is usually assumed that demand is deterministic and invariant to airline changes and competition. However, this assumption leads to overestimates in the number of passengers served. In particular, the paper studied the effect of competition with high-speed rail services. The authors developed two models: one exclusively for market demand (Model 1) and another for scheduling (Model 2). Model 1 uses a nested logit model with a range of parameters suitably defined to estimate the influence of competitors. The estimation takes place using maximum likelihood estimation and a historic data set from a Spanish airline, IBERIA, and high-speed rail services. Model 2 is an integrated model for frequency planning, approximate timetable development, and fleet assignment. It uses a nonlinear MIP formulation with dynamic time discrete intervals (number of time periods considered are lower for uncongested airports). The formulation takes the results from the market demand, Model 1, as input parameters. Model 2, therefore, takes airport slots availabilities, fleet size, average fares, demand, and competitors schedules as inputs. The authors referred to Model 2 as the integrated airline scheduling under competition model. The case study considered data from IBERIA (hub-and-spoke network) with 23 airports and 104 routes. Using a standard computer, optimal solutions were found (with at most 1% optimality gap) in under 2 seconds. Authors also included an assessment using standard statistics (F-tests, t-tests, and hypothesis testing) that assessed the fit of Model 1. The schedules produced by Model 2 were found to be

reasonably close to the original decisions made by the airline.

The fact that fuel costs are a driving factor in airlines' expenditure motivated the authors in Gürkan et al. (2016). In this paper, the authors proposed an operational and robust mixed integer nonlinear formulation that integrated flight scheduling, fleet assignment, and aircraft routing with special attention on cruise speed control for the fuel consumption element. The type of aircraft used and duration of trips were considered. To remove the non-linearity introduced by the fuel consumption formula, authors presented a conic reformulation. Authors developed a discrete approximation and cruise speed control algorithm (Heuristic 1), and a multi-stage triplet search algorithm (Heuristic 2). A computational study revealed that, on average, Heuristic 1 provided better solutions while Heuristic 2 was faster.

Integrated Airline Recovery

When compared to its pre-operational equivalent, the integrated airline recovery literature is more limited. Commonly, the objective function is to minimise the so-called recovery cost. For a given stage, the recovery cost may include costs for delays, cancellations, and additional penalties. The total recovery cost for a given problem, is the sum of the recovery costs incurred by the individual stages.

The first approach that integrated all stages of the airline recovery problem was proposed by Lettovsky (1997). In practice, however, the author only applied the framework to the crew recovery problem (Lettovsky et al., 2000).

Abdelghany et al. (2008) published a decision support tool for airlines schedule recovery that allowed the integration of aircraft and crew recovery. The approach,

which consists of several interacting components including preprocessing, simulation, and a “greedy optimisation strategy” was run on a rolling horizon basis. The simulation component accounted for delay propagation of a given disruption across a flight network, hence, predicted the disrupted flights. The greedy optimisation strategy, based on an efficient MIP, provided solutions for the semi-integrated airline recovery problem. The objective function minimises the recovery cost which includes cancellations, delays, and penalties: for deviation from the original schedule (cost of swapping aircraft), idle and standby crew and aircraft, and deadhead flights. The test instances considered, 1100 flights and 552 aircraft, were solved in reasonable computational times.

Petersen et al. (2012) were the first to implement a framework that integrated all components for an integrated airline recovery. Closely related to the work by Letovsky (1997), authors introduced a model that integrated all problems. A solution procedure was implemented that provided a balance between solving the fully integrated model and tractability. Their procedure, which revolved around some custom preprocessing and an efficient solution algorithm, provided globally optimal solution to the flight and passenger recovery problems. To limit the scope of the problem, in an attempt to make it tractable, authors proposed two main preprocessing steps. First, the identification of what authors referred to as disruptable flights. These are, given a disruption (e.g. airport closure), flights that are either directly affected or are candidates for disruption. The latter, are identified as flights that are not directly affected by the disruption, but their delay/cancellation is likely to be beneficial for the whole recovery process. Second, to create flight delay copies authors introduced

event-driven delays. Given a maximum delay window and some airport constraints (e.g. gate or slot restrictions), flight delay copies are generated to avoid these restrictions. Hence, significantly limiting the number of copies for a given flight. The solution algorithm, an iterative Benders' decomposition algorithm, solved the passenger and flight recovery problems to optimality. For the other components, aircraft and crew, independent LP-relaxations were solved, on which branching is performed to achieve integrality. The objective functions for each stage, minimise the respective recovery cost; these include cancellations (flights and passengers), delays (passengers), and penalties for deadheading (flights and crew). The computational tests presented, for a US-based carrier, daily recovery solutions for 800 flights in under 30 minutes. Moreover, the authors introduced a combined Benders' decomposition and column generation algorithm (column-and-row generation) for the crew recovery problem.

Maher (2016) presented a solution algorithm that solved the semi-integrated airline recovery problem (flight, aircraft, and crew). Here, the recovery cost includes flight cancellations and delays, and penalties for crew (deadheading, extra remuneration for longer recovery shift and use of reserve crew). The solution algorithm extended the generic column-and-row generation, by Muter et al. (2013), which enabled its efficient application on the semi-integrated airline recovery formulation. For the computational tests, different disruption scenarios were generated using two test instances with different network structures. A central hub instance (262 flights, 48 aircraft), and hub-and-spoke instance (441 flights, 123 aircraft). Using the proposed solution approach, computational times shown were an average of 427 seconds for the central hub instances and 400 seconds for the hub-and-spoke instances. Moreover, tests report

the computational benefit of the column-and-row generation algorithm over a column generation algorithm. This framework was extended by the same author to account for passenger recovery in Maher (2015). Hence, providing a solution procedure for the fully integrated airline recovery problem.

Marla et al. (2016) proposed a semi-integrated (aircraft and crew recovery) TSN-based model that, additionally, considered a new step: flight planning. This additional step, which involved the calculation flight trajectories, allowed the consideration of several supplementary factors in the decision making process; the most influential one being fuel burn. Authors included flight planning by generating flight copies at each departure time for each alternative cruise speed. The evaluation of these flight copies is then outsourced to a separate tool, which provides information about the resulting fuel burn and other flight specific details. The results are combined in the scenario generation with a passenger delay simulator, as an input to the formulation of the approximate semi-integrated formulation. These parameters are employed to calculate the combined objective which includes recovery and fuel costs. Recovery costs account for cancellations (flights and passengers), delays (flights and passengers), and penalties (aircraft swaps and incremental delay). The computational tests, for an EU-based airline with 250 daily flights were solved with a time limit of 2 minutes, hence, the problem was not solved to optimality.

Arıkan et al. (2017), extending previous work by Arıkan et al. (2016), presented a semi-integrated (aircraft, crew and passenger recovery) modelling approach which also included some flight planning considerations. Particularly, authors accounted for the effect of cruise speed by modelling its effects on the fuel costs. Due to the fuels

costs constraints being nonlinear, the authors formulated the problem using a conic quadratic mixed integer programming formulation. Similarly as with Marla et al. (2016), the objective function minimises the combined recovery and fuel costs. Recovery costs include cancellations (flights), delays (flights and aircraft), and penalties (airport closure). Preprocessing routines are included to ensure problem sizes are manageable. These ensure that the proposed flight network representations (similar to those introduced by Desaulniers et al., 1997) remain manageable and the scope of the recovery is limited. Computational tests revealed that reasonable test instances (288 flights) can be solved within 10 minutes.

Alternative Approaches for Airline Scheduling and Recovery

Some more infrequent models for the different stages of airline scheduling throughout the literature can be found in this section. Specifically, discussed here is the modelling of passenger recovery through a multiplex network (Cardillo et al., 2013); semi-integration of flight scheduling and fleet assignment through robust multi-objective optimisation (Burke et al., 2010); and crew scheduling through Potts mean field techniques (Lagerholm et al., 2000).

By modelling a network from European Air Transport Network using a multiplex graph and an algorithm for re-scheduling under random failures (disturbances), Cardillo et al. (2013) developed a robust passenger recovery model. The idea of a multiplex graph in this context represents the operations of different airlines as a network on multiple layers (where each layer represents one airline). Superposition of networks gives an accurate visualisation of the air traffic busiest airports as a whole and also

in the individual layers. The multiplex network is constructed by the summation of a layer-measure for the number of edges incident to a vertex, providing a global degree for each vertex. The layer-measure allows for the calculation of the probability that a given terminal will be selected as origin or destination. This is set to be the ratio of the global degree of the vertex over the sum of the global degree over all the vertices in the multiplex graph. For the inclusion of a random failure, the authors introduced a hopping distance which is simply the minimum distance between two airports across all the layers (disregarding the airline). To model disturbances, a random set of edges is removed. Authors outlined an algorithm for re-scheduling by using current active paths and re-routing passengers to different flights and if necessary, airlines (satisfying a load constraint), until all passengers complete their initial journey.

There are two central methods in the optimisation framework to address uncertainty: **stochastic programming** and **robust optimisation**. Stochastic programming requires alternating scenarios for uncertain data by drawing from appropriate distributions. However, if the distributions are unknown then there is a challenge to estimate and assess the fit on the data. Robust optimisation, on the other hand, only requires the uncertain quantity of interest to lay between some known intervals. By such a deterministic management of uncertainty, this approach focuses on providing a robust schedule. In the airline scheduling context, models of this type considerably reduce the number of build-up delays by making the schedule stable against disturbances.

Burke et al. (2010) applied robust optimisation with multiple objectives to solve

a semi-integrated model for the flight scheduling and fleet assignment problems. Authors argued that the simultaneous consideration of these, within a robust optimisation setting, provides schedules that lead to a better operational performance, also accounting for an optimal trade-off between the multiple objectives according to the priority attributed to these. For the solution algorithm, authors applied a multi-meme memetic algorithm, a hybridisation of the genetic algorithm with local search. It is initialised, either randomly or by a biased set of criteria, and then produces an approximate Pareto front which is improved upon in consequent iterations. Authors revealed that, the flexibility provided by the robust objectives at the operational level ensures good quality solutions for longer term schedules. The computational tests used a relatively small data set from the KLM Royal Dutch Airlines and the University of Nottingham's high-performance computing facility to run the experiments.

Lee et al. (2006) modelled the flight scheduling problem as a multi-objective programming problem, which was solved using a multi-objective genetic algorithm. To make the solution robust, airlines selected the initial values for the parameters representing the measures of robustness (e.g. perturbation allowed for flight departure times), these were then optimised according to the objectives. The computational times for the algorithm were around 90 hours on an outdated computer.

Several artificial intelligence approaches can also be applied to the airline scheduling problem and other resource allocation problems. Lagerholm et al. (2000) proposed the use of artificial neural networks for the modelling of the crew scheduling problem. Specifically, authors developed an approach based on the so-called mean field Potts approach. Utilising the fact that feedback artificial neural network methods can be

applied to solve set partitioning problems, and since the crew scheduling problem may be transformed into a set partitioning problem, an algorithm is presented for this problem. The algorithm is based on first, narrowing down the solution space by using an advanced reduction technique that removes a large part of the suboptimal solutions, and then, using a mean field approach based on a Potts neuron encoding. They highlighted the benefits and applications to other complex combinatorial problems. A computational case study was presented showing that the crew scheduling problem can be solved on a large flight network (1000 flights and 50 airports) in just over 10 minutes.

2.4 Methodology for Chapter 4

This section contains the background required for the methodology used in Chapter 4. Explicitly, we include, a well-known mathematical optimisation solution algorithm, **column generation** 2.4.1; a brief overview of **heuristics** 2.4.2; and the **shortest path problem with resource constraints** (SPPRC) 2.4.3, a useful type of subproblem in the column generation framework.

2.4.1 Column Generation

Introduced by Dantzig and Wolfe (1960), Dantzig-Wolfe decomposition or column generation tackles large problems by iteratively generating variables until the optimal solution is reached. This is done by breaking down the original problem into a simple master problem and K subproblems. Hence, when a problem has a large number

of variables, making it intractable to solve using standard methods, one can employ column generation. Nevertheless, the applicability and efficacy of the method relies on two assumptions, the constraint matrix can be transformed to possess a specific form and the primal simplex method can be applied to re-optimize the master problem in every iteration.

Consider the following standard LP formulation,

Model 2.4.1. *Standard LP formulation.*

$$\min_x \quad \mathbf{c}^T \mathbf{x} \quad (2.4.1a)$$

$$s.t. \quad \mathbf{A}\mathbf{x} = \mathbf{b} \quad (2.4.1b)$$

$$\mathbf{x} \geq 0 \quad (2.4.1c)$$

where $x = (x_1, \dots, x_K)^T$ and $b = (b_1, \dots, b_K)^T$. In order to apply Dantzig-Wolfe decomposition, the constraint matrix, A , has to retain a specific form. Let B_k and A_k for $k = 1, \dots, K$ be submatrices with appropriate dimensions, then, A has a special structure as depicted below,

$$\begin{pmatrix} B_0 & B_1 & B_2 & \dots & B_K \\ & A_1 & & & \\ & & A_2 & & \\ & & & \ddots & \\ & & & & A_K \end{pmatrix}.$$

This structure, called the block angular structure, provides the necessary submatrices to reformulate the problem into a master problem and K subproblems. The top row, which corresponds to the constraints of the master problem, i.e. $\sum_{k=1}^K B_k x_k = b_0$, are referred to as the coupling constraints. The subproblems, with constraints of the form

$A_k x_k = b_k$ for $k = 1, \dots, K$; do not all have to be solved.

To reformulate the master problem, Minkowski's representation theorem can be employed. The theorem states that extreme points or rays are represented by a set of points which lay on the boundary of a convex region.

Theorem 2.4.1 (Minkowski's representation theorem, Minkowski, 1953). *If $P = \{\mathbf{x} : A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0\}$ is a feasible (bounded or unbounded) region, then we can represent any point $\mathbf{x} \in P$ as a linear combination of its extreme points or rays, $x^{(j)}$,*

$$\mathbf{x} = \sum_j \lambda_j x^{(j)} \quad \text{as long as} \quad \sum_j \delta_j \lambda_j = 1 \quad \text{and} \quad \lambda_j \geq 0$$

where

$$\delta_j = \begin{cases} 1, & \text{if } x^{(j)} \text{ is an extreme point (case that } P \text{ is bounded)} \\ 0, & \text{if } x^{(j)} \text{ is an extreme ray (case that } P \text{ is unbounded)}. \end{cases}$$

Hereby, each column of the master problem can be made to represent a solution, as a linear combination of extreme points and rays, of each of the subproblems. The model can be reformulated as follows,

Model 2.4.2. *Standard LP Reformulation using Theorem 2.4.1.*

$$\min_{x_0, \lambda_{kj}} \quad c_0^T x_0 + \sum_{k=1}^K \sum_{j=1}^{p_k} \left(c_k^T x_k^{(j)} \right) \lambda_{kj} \quad (2.4.2a)$$

$$s.t. \quad B_0 x_0 + \sum_{k=1}^K \sum_{j=1}^{p_k} \left(B_k x_k^{(j)} \right) \lambda_{kj} = b_0 \quad (2.4.2b)$$

$$\sum_{j=1}^{p_k} \delta_{kj} \lambda_{kj} = 1 \quad k = 1, \dots, K \quad (2.4.2c)$$

$$x_0, \lambda_{kj} \geq 0 \quad (2.4.2d)$$

In practice, this reformulation can not be applied directly, as the number of λ -variables becomes very large because the number of extreme points and rays for every subproblem is also large. However, the fact that many of these variable will end up being non-basic may be exploited by the column generation algorithm. So that, only the variables with a negative reduced cost, therefore likely to become basic, are explored. With this consideration, the restricted master problem (RMP) may be reformulated as,

Model 2.4.3. *Restricted Master Problem.*

$$\min_{x_0, \lambda'} \quad c_0^T x_0 + \mathbf{c}^T \lambda' \quad (2.4.3a)$$

$$s.t. \quad B_0 x_0 + B \lambda' = b_0 \quad (2.4.3b)$$

$$\Delta \lambda' = 1 \quad (2.4.3c)$$

$$x_0, \lambda' \geq 0. \quad (2.4.3d)$$

The dimensions of this problem evolve as variables are added when the subproblems are solved. If a subproblem k has been solved, the variable $\lambda_{k,j}$ can be included if its reduced cost is negative. The master problem is extended by introducing the column and its corresponding cost coefficient.

Let the dual variables for constraints 2.4.3b and 2.4.3c be denoted by π_1 and $\pi_2^{(k)}$

respectively. To determine which new column should be introduced to the master problem, a reduced cost LP needs to be solved for each subproblem under consideration. Assuming the subproblem k is bounded, the reduced cost LP gives the K formulations for the subproblems as,

Model 2.4.4. *Reduced cost LP.*

$$\min_{x_k} \quad \sigma_k = (c_k^T - \pi_1^T B_k) x_k - \pi_2^{(k)} \quad (2.4.4a)$$

$$s.t. \quad B_k x_k = b_k \quad (2.4.4b)$$

$$x_k \geq 0. \quad (2.4.4c)$$

Now, the most significant x_k to enter the master problem are those found to maximise the reduced cost LP. So, if $\sigma_k^* < 0$ then the new column, $\lambda_{k,j}$, can be introduced to the master problem with its corresponding cost coefficient, $c_k^T x_k^*$.

The pseudo-code for the column generation algorithm is given in Algorithm 1. It consists on iteratively solving the RMP, and one of the K subproblems at a time. The subproblem is solved by solving the corresponding reduced cost LP. Hence, if, after solving the k -th subproblem, we see that objective is negative, then, the corresponding variable has a negative reduced cost, and we generate and add the column to the RMP. We repeat the process, moving on to the next subproblem until no negative reduced cost is obtained. Moreover, the initialisation involves checking the subproblems and generating the first columns. If any of the subproblems happens to be infeasible, then the master problem is also infeasible. Otherwise, an initial feasible solution can be easily constructed from the optimal solutions to these.

Algorithm 1 Column Generation Algorithm.

```

1: INPUTS: RPM (Model 2.4.3),  $K$  subproblems (Model 2.4.4)
2: Initialisations: stop = False.
3: while stop is False do
4:   Solve RPM;
5:   for  $k = 1, \dots, K$  do
6:     Solve subproblem  $k$ 
7:     if  $\sigma_k^* < 0$  then
8:       Update the restricted master problem by adding proposal  $x_k^*$ .
9:       Introduce column  $\begin{pmatrix} c_k^T x_k^* \\ B_k x_k^* \end{pmatrix}$ .
10:    end if
11:  end for
12:  if No proposal generated then
13:    stop = True. ▷ Optimality has been reached.
14:  end if
15: end while

```

This algorithm is extremely versatile and has been used to solve multiple types of problems. However, if the original problem is a MIP and optimal solutions are sought, column generation has to be embedded in a branching algorithm to form the so-called branch-and-price algorithm (Barnhart, Johnson, Nemhauser, Savelsbergh and Vance, 1998).

2.4.2 Heuristics

In what follows, we discuss, in some level of generality, different methods and algorithms that can be applied to a wide variety of problems with the aim of providing good approximations in short computational times. These methods are called **heuristics**. More complex and generic heuristics have been developed with a range of different themes, inspirations and variations, these are called **metaheuristics**. Lastly, for an increasing variety of problems, the emerging field of **hyper-heuristics**, exploits the

use, of one or both of these methods to provide adaptive approximations.

Introduction

The motivation behind the use of heuristics comes from the cases when finding an exact solution to a problem is computationally costly. This may occur due to, for example, the number of solutions being extremely large, so a long time is spent exhaustively examining different solutions and selecting the optimal ones. Additionally, and in other cases, obtaining feasible solutions may be in itself the bottleneck. Heuristic methods aim provide approximate solutions fast. On the other hand, heuristic methods sacrifice optimality for the computational gains and, in some cases, they are only marginally faster than exact algorithms. Thus, this trade-off, must be taken into account.

The development of heuristic algorithms typically involves some “rules of thumb” applicable to a specific problem. Heuristic principles or dictionaries, were introduced in the widely influential book by Polya (1945). The idea behind these principles is to help exploit the structure of the problem, and some of its properties, to obtain a simplification that can be solved with an approximate method. However, such simplification should be close enough to the initial problem for the solution to be meaningful, and differ enough so that it can be solved easily. The following famous principles (from Polya, 1945), help to achieve this balance,

analogy: mapping the problem to an analogous problem;

auxiliary elements and problems: adding an element that will make the prob-

lem easier, and finding subproblems;

backward reasoning: working backwards from a known solution/hypothesis;

decomposing and recombining: decomposing the problem into more manageable parts and recombining them;

generalisation: finding a generalisation to the original problem;

induction: deriving a generalisation from some simpler cases;

specialisation: finding a specialised variant of the problem.

Based on these principles, more complex and generic heuristics were developed. The field evolved into what is now known as *metaheuristics*; the name first appeared in Glover (1986).

Metaheuristics

There are two categories of metaheuristics: **trajectory-based** and **population-based** (Gendreau and Potvin, 2005).

The simplest kind of trajectory-based heuristic is local search (a.k.a. neighbourhood search or hill-climbing). Local search begins by constructing a single feasible solution, and then iteratively makes small improvements to it, until no further improvements can be found. More complex trajectory-based approaches contain special mechanisms to enable the search to escape from local optima. Prominent examples include Simulated Annealing (introduced and applied to the travelling salesmen problem by Kirkpatrick et al., 1983), Tabu Search (inspired by Glover, 1977, later

formalised by Glover, 1989), and Greedy Randomised Adaptive Search Procedure (GRASP, introduced by Feo and Resende, 1995). For an example application and implementation of some trajectory-based metaheuristics see Chapter 4.

Population-based algorithms, as opposed to finding a single solution, find a range (population) of independent solutions which can be combined to create different solutions, effectively guiding the algorithm towards better solutions. Some of the most popular population-based metaheuristics are nature-inspired algorithms. These include Ant-Colony Optimisation (introduced by Dorigo, 1992, popularised by Dorigo et al., 2006), Particle Swarm Optimisation (Kennedy and Eberhart, 1995), and Genetic Algorithms (Holland, 1975). However, more recently, a vast amount of algorithms from this stream have appeared. From the African Buffalo (applied to the TSP by Odili et al., 2016), through Moth-Flame Optimisation (applied to several engineering problems by Mirjalili, 2015), all the way to Zombie Optimisation (applied to pedestrian image tracking by Nguyen and Bhanu, 2012).¹

Hyper-heuristics

Hyper-heuristic search methods incorporate different machine learning techniques to automate the solution of an evolving set of optimisation problems, hence, being able to adapt to the challenges in a suitable fashion to produce promising solutions. With a set of low-level heuristics or metaheuristics, hyper-heuristic methods exploit the **combination** or **selection** of these to generate improved solutions.

Combination or generation hyper-heuristics combine basic components of dif-

¹A full and updated list of nature-inspired and metaphor-containing metaheuristics can be found at: <https://github.com/fcampelo/EC-bestiary>.

ferent heuristic methods to construct new heuristics. Heuristic methods frequently consist of problem-dependent simple local search operations. For example, in a generic scheduling setting, some simple local search operations, that add activities to a schedule under construction, could be: add the activity with longest/shortest processing time, add a random activity, try all the possible candidate activities (not already present in the current schedule) and add the one that best suites the current schedule (e.g. minimises a certain cost function, or satisfies a certain constraint).

Figure 2.6 shows a scheme for a **selection** hyper-heuristic. Given a problem specific input and three different heuristics(or metaheuristics) H1, H2, H3, in each iteration the hyper-heuristic algorithm selects one heuristic (based on the ranking), applies it on the problem under consideration, evaluates its performance, and updates the ranks of all heuristics based on the performance.

Hyper-heuristic methods model an interaction between statistics and operational research, and have received attention from researchers from both communities. This has led to a wide array of applications including (and not limited to) scheduling problems, e.g. educational timetabling (Burke et al., 2007), nurse rostering (Burke et al., 2003); the travelling salesman problem (Keller and Poli, 2007; Runka, 2009); vehicle routing problems (Ochoa et al., 2012); and knapsack problems, mutidimensional (Allen et al., 2009), and 0/1 (Kumar et al., 2008). Furthermore, there are recent theoretical publications that study computational complexity of different methods, Doerr et al. (2018), for instance. Given the scope of the area, we refer the reader to the recent and thorough review by Burke et al. (2013), and Burke et al. (2019). In Chapter 4 we employ a learning-based hyper-heuristic to adaptively select different

metaheuristic algorithms in a column generation framework.

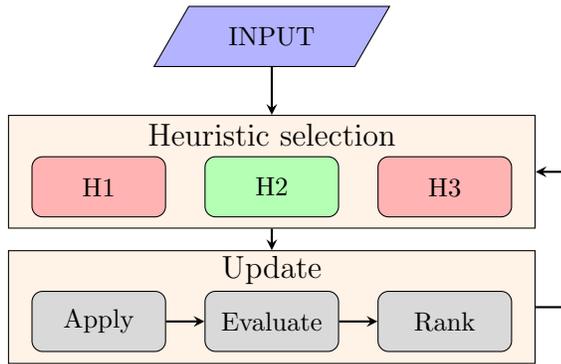


Figure 2.6: Hyper-heuristic algorithm.

2.4.3 Shortest Path Problem with Resource Constraints

In the applied column generation framework, particularly in scheduling related literature, the shortest path problem with resource constraints (SPPRC) is commonly employed to generate variables as a subproblem. It consists, as its name suggests, in finding, among all paths, the shortest path from source to sink nodes, that satisfies a set of constraints for a defined set of resources.

The SPPRC was first introduced by Desrochers (1986) as a subproblem for the bus driver scheduling problem. It has been widely applied in a variety of different settings including: the vehicle routing problem with time windows, the technician routing and scheduling problem, the capacitated arc-routing problem, on-demand transportation systems, and airport ground movement (Desrochers and Soumis, 1988; Feillet et al., 2004; Irnich and Villeneuve, 2006; Righini and Salani, 2008; Bode and Irnich, 2014; Chen et al., 2016; Garaix et al., 2010; Tilk et al., 2017; Zamorano and Stolletz, 2017).

To discuss the SPPRC in more detail, let us define some notation. Let $G = (N, A)$ denote a directed graph with set of nodes N and arcs A . Let $\mathcal{R} = \{1, \dots, R\}$ denote

the set of resources, and $\underline{\mathbf{L}} = (\underline{L}^1, \dots, \underline{L}^R)$ and $\overline{\mathbf{L}} = (\overline{L}^1, \dots, \overline{L}^R)$ denote vectors for minimum and maximum resources, respectively. For each arc i , we denote the associated weight by w_i , and the vector of resource consumption by $\mathbf{f}_i = (f_i^1, \dots, f_i^R)$. Each component in the vector is called a resource extension function (REF) (Irnich and Desaulniers, 2005). For an example of how these can be applied see Section 4.3.3.

For a given path, p , between a given pair of nodes, we denote the set of connections by $A(p)$, the total weight of the path by $w(p)$, and the resource consumed along the path by $\mathbf{f}(p) = (f^1(p), \dots, f^R(p))$. Where,

$$w(p) = \sum_{i \in A(p)} w_i \ ; \ f^r(p) = \sum_{i \in A(p)} f_i^r .$$

The path is said to be resource feasible if $\underline{\mathbf{L}} \leq \mathbf{f}(p) \leq \overline{\mathbf{L}}$, i.e. $\underline{L}^r \leq f^r(p) \leq \overline{L}^r$ for every resource r .

Solution Algorithms

There are a number of algorithms for the standard shortest path problems. Among some of the most historically relevant algorithms are greedy algorithms, **Dijkstra's** algorithm (Dijkstra et al., 1959) (which is restricted to graphs with non-negative edge weights) and **Bellman-Ford** algorithm (Bellman, 1958; Ford Jr, 1956), and best-first search, **A* search** algorithm (Hart et al., 1968) (which uses a heuristic function to provide faster solutions). Nevertheless, these algorithms cannot be used directly to solve the SPPRC. The reason for this is the need to compare the resources consumed when obtaining the different paths. This is done through the use of labelling algorithms. Simply put, labels carry information about the different resources under

consideration, allowing comparisons between different potential paths. Alternatively, as introduced in Chapter 4, standard algorithms can be employed to provide approximate but fast solutions for the SPPRC. This is done by iteratively solving a standard shortest path problem and modifying the graph so as to force resource feasibility.

Many **exact** algorithms to solve the SPPRC are dynamic programming (DP) labelling algorithms. These build paths in a systematic fashion by starting at a source node and traversing the graph considering all feasible directions. Labels are created in order to efficiently compare different paths and discard the dominated or “suboptimal” ones.

Irnich and Desaulniers (2005) presented an exact DP algorithm, the **monodirectional forward labelling** algorithm (sometimes called unidirectional), based on the pioneering work by Desrochers and Soumis (1988). Boland et al. (2006) published a state augmenting algorithm that used a monodirectional labelling algorithm to find an elementary path (one without repeating nodes). Righini and Salani (2006) introduced a **bidirectional labelling** algorithm for the SPPRC. The bidirectional algorithm is an extension of the monodirectional algorithm that supports search from both ends of the graph, hence reducing the computational efforts. Moreover, they used bounding in order to mitigate label explosion. This typical phenomenon occurs simply due to the nature of the algorithm and the need to store the non-dominated labels of all partial paths. More recently, Tilk et al. (2017) released a bidirectional labelling algorithm with dynamic halfway point. Based on previous works (Righini and Salani, 2006; Pecin et al., 2017), the bidirectional search is bounded for both directions and

these bounds are dynamically updated as the search in either direction advances. For more details and an implementation of this algorithm please see Section 4.4.3.

Some **heuristic** algorithms to solve the SPPRC are also based on DP. Additionally, **metaheuristic** algorithms have been developed to find fast and promising solutions to the SPPRC. DP heuristics have been developed by several authors (Feillet et al., 2004; Lozano et al., 2015). Local search or metaheuristics, in this setting, start with a given path and perform a series of moves (node/arc: deletion, insertion, or exchange) to obtain another feasible path with lower cost. Metaheuristics implemented for the SPPRC include trajectory-based, Tabu search (Desaulniers et al., 2008) and GRASP (Ferone et al., 2019), and nature-inspired, hybrid Particle Swarm algorithm (Marinakis et al., 2017). For more information on metaheuristic algorithms for the SPPRC, including two new algorithms, see Section 4.4.3. For a Python implementation of the hybrid Particle Swarm algorithm see Torres Sanchez (2020).

We now proceed with a discussion the basic labelling algorithm.

Monodirectional Forward Labelling Algorithm

Algorithm 2 shows a generic version of the monodirectional forward labelling algorithm (Irnich and Desaulniers, 2005). It uses a set of unprocessed paths U and a set of processed paths P , which change dynamically with every iteration. After an appropriate initialisation, in line 3, an unprocessed path is selected, say Q and subsequently removed from the set of unprocessed paths, line 4. In lines 5 to 10, all the feasible one node extensions are computed and checked for resource feasibility. In lines 7 and 8, if the extension is resource feasible, then it is added to the set of unprocessed paths

U , and the original path is added to the set of processed paths P . After this, if we wish to eliminate some suboptimal paths, dominance relations can be applied to both unprocessed and processed paths.

Algorithm 2 Generic monodirectional labelling algorithm for the SPPRC.

```
1: INPUTS:  $G = (V, A)$ 
2:  $U = \{(s)\}$  and  $P = \emptyset$ ; ▷ Initialisation
3: while  $U \neq \emptyset$  do
4:   Choose a path  $Q \in U$ , with  $h(Q) = i$ ; remove  $Q$  from  $U$ ;
5:   for  $(i, j) \in A$  do
6:     Extend path  $Q$  along  $(i, j)$ , denoted by  $(Q, j)$ ;
7:     if  $(Q, j)$  is feasible then
8:       Add  $(Q, j)$  to  $U$ ; add  $Q$  to  $P$ ;
9:     end if
10:  end for
11:  Apply dominance rules to  $U \cup P$ ;
12: end while
```

Chapter 3

An Optimisation Framework for

Airline Fleet Maintenance

Scheduling with Tail Assignment

Considerations

3.1 Introduction

There are a number of operational decisions associated with airlines, from ticket prices to flight times, crew rosters, and aircraft maintenance. When making these decisions, airlines have to take into account their own economic interests influenced by demand, costs, and sometimes even the actions of their competitors. In such a competitive environment, airlines aim to minimise their operating costs while providing competitive services. Significant proportion of operating costs are dedicated to maintenance. For

instance, 20.5% of the average direct operating cost per medium-haul trip are dedicated to maintenance on an Airbus A330-200 (Aircraft Analysis & Fleet Planning, 2005). Therefore, it is of paramount importance to develop decision making tools that will allow airlines to optimise their aircraft maintenance decisions.

Maintenance types are classified according to: short, medium and long-term interventions. Short-term or line maintenance does not require modelling or advanced planning as they are carried out as standard procedures at airport gates. Medium and long-term maintenance interventions include:

1. Airframe checks (A, B, C and D);
2. Engine performance restoration (EPR) and life limited parts replacement (ELR);
3. Landing gear overhaul (LG), and;
4. Auxiliary power unit (APU) performance restoration.

Civil Aviation Authorities Regulations impose that maintenance has to be performed after a certain number of months (MO), flying hours (FH), or flight cycles (FC), at certified maintenance workshops. In the medium-term, A checks have to be performed every 80-100 FH (every 7 to 9 days), requiring 10-20 man-hours, while B checks typically occur every 500-600 FH (every two months), requiring 100-300 man-hours (Department for BIS, 2016). However, in practice, Type B checks are included as part of a longer A check, or a bundle of A checks (Qantas, 2016). Long-term maintenance, including C and D checks, LG, EPR, ELR, and APU are performed once every 1–6 years and can last over 10 days (Ackert, 2011). It is worth noting that

there is a large variability in the duration of maintenance checks due to the fact that different aircraft types have different maintenance requirements.

Table 3.1 shows the frequency of the four airframe checks for various aircraft types. As one would expect, the time between checks increases for more modern aircraft and B checks disappear, being contained in longer A checks. Maintenance is performed before any of the three criteria (MO, FH or FC) is met. For instance, for the B737-200, a C check is performed after 18 MO, 6000 FH, or 3000 FC, whichever occurs first. The justification for this practice is to ensure coverage of overused aircraft operating short-haul flights. In these cases, FC are accumulated faster than FH (Cook and Tanner, 2008). Moreover, usual frequencies of long-term maintenance, for the A320, for instance, are 13500 FC for both the EPR and the ELR, 120 MO/20000 FC for the LG, and 75000 FH for the APU (Ackert, 2011).

Table 3.1: Typical maintenance frequencies in calendar months (MO), flying hours (FH), or flight cycles (FC) (Cook and Tanner, 2008; Martins, 2016).

Aircraft	A check	B check	C check	D check
B737-300	275 FH	825 FH	18 MO	48 MO
B737-400	275 FH	825 FH	18 MO	48 MO
B737-500	275 FH	825 FH	18 MO	48 MO
B737-800	500 FH	n/a	4000-6000 FH	96-144 MO
B757-200	500-600 FH	n/a	18 MO/6000 FH/3000 FC	72 MO
F100	500 FH	n/a	5000 FH	12000 FH
B767-300ER	600 FH	n/a	18 MO/6000 FH	72 MO
B747-400	600 FH	n/a	18 MO/7500 FH	72 MO
A319	600 FH	n/a	18-20 MO/6000 FH/3000 FC	72 MO
A320	600 FH	n/a	18-20 MO/6000 FH/3000 FC	72 MO

For maintenance to be performed effectively, there is an essential underlying process, airline fleet maintenance scheduling (AMS). The AMS problem deals with the construction of a schedule that minimises maintenance costs, resource usage, and the

disruption to airline operations, while satisfying current safety regulations by different civil aviation authorities (Sriram and Haghani, 2003). Given the frequency of the maintenance checks, the decision horizon for medium-term maintenance should be a month, and, at least, six months for long-term maintenance.

A challenge for the AMS problem is to allocate maintenance-related resources in a cost-effective fashion. These resources are geographically dispersed throughout distant and distinct maintenance workshops. Some examples of resources include limited specialised tools, spare parts, and certified technicians. Additionally, regulated checks depend on the state of the aircraft and employ different resources.

The major contribution of this paper is the framework that deals with the requirements introduced by 30-day planning horizon instances, with multiple airlines and workshops, and tight resource availabilities. Such framework can be broken down into three steps. Firstly, the preprocessing step identifies maintenance opportunities (MOPs). These opportunities arise when flights have large turnaround times during which aircraft can be maintained. This preprocessing allows us to formulate and solve the problem efficiently. Next, in the case when not enough MOPs are found and maintenance regulations are violated, we identify when these occur and we reassign aircraft to different flights to generate more MOPs. In other words, we re-solve the tail assignment (TA) problem. Lastly, to preserve tractability and improve the quality of solutions, we provide a two-stage iterative algorithm. The first stage selects a conflicting period which identifies the smallest number of flights whose reassignment might lead to a maintenance feasible solution. Such period starts from the mainte-

nance regulation violation, and ends at the next available MOP. The second stage makes the timeline more granular to improve resource allocation. Further contributions introduced by this work are: the presentation of a new type of multi-objective optimisation formulation for the AMS which copes with single and multi-workshop cases, the focus on workshop resource allocation, the consideration of different fleet types and their respective maintenance requirements, and, the introduction of a solution framework that solves problems of realistic size in short time. Showing the potential of this framework which promotes cooperation between airlines.

The remainder of this paper is organised as follows: Section 3.2, discusses the relevant literature. Section 3.3 presents the proposed modelling approaches and the corresponding underlying concepts. The first model, minimises the number of maintenance regulation violations, adhering to the criteria presented in Table 3.1. The second model solves a reduced TA problem within the AMS considering violations. Section 3.4 presents the solution methodology and the model application. Computational experiments are given in Section 3.5, while Section 3.6 summarises the conclusions and provides recommendations for future research.

3.2 Literature Review

The airline planning process or airline scheduling problem involves several stages: flight scheduling, sometimes referred to as schedule design; fleet assignment, which assigns fleet types to flights (Hane et al., 1993); TA, sometimes called aircraft routing or aircraft rotation and involves assigning individual aircraft to flights (Clarke et al., 1997); maintenance scheduling (MS); and, finally, crew scheduling. These were tradi-

tionally modelled by formulating each stage separately and solving them sequentially, i.e. the output of one stage is the input for the next. Figure 3.1 presents the order of solving the different types of problems identified above (solid lines), along with some common feedback loops (dashed lines) and the associated typical planning horizons.

The sequential modelling and solution of the airline scheduling problem does not take into consideration the restrictions of the subsequent problems. The benefit of the sequential approach is the reduction in computational complexity. Even though the sequential feedback system is a close approximation, the solution can be improved by modelling the interdependence of each stage in an integrated model (Cordeau et al., 2001). Integrated models have been developed to provide better quality results and consider the combination of two or more stages into a single problem (Desaulniers et al., 1997; Clarke et al., 1997; Barnhart, Boland, Clarke, Johnson, Nemhauser and Shenoi, 1998; Cohn and Barnhart, 2003; Sriram and Haghani, 2003; Mercier et al., 2005; Sarac et al., 2006; Liang and Chaovalitwongse, 2013; Safaei and Jardine, 2018). In particular, MS is frequently contained within the TA, in which case it is called aircraft maintenance routing (AMR) problem (Gopalan and Talluri, 1998).

The most common types of mixed integer programming formulations for the integrated airline scheduling problem can be classified into three groups,

String-based models: a type of formulation that models the problem using strings,

i.e. sequences of connected flights that begin and end at a maintenance workshop, and, that satisfy flow balance and maintenance regulations (Desaulniers et al., 1997).

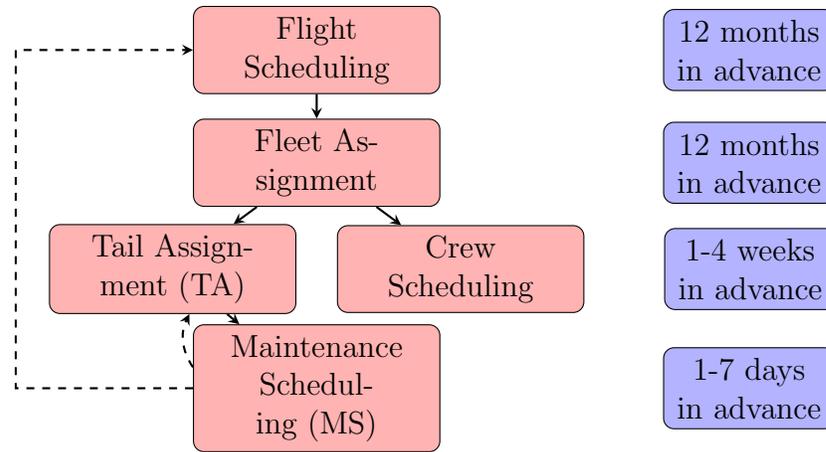


Figure 3.1: Stages of the airline scheduling problem.

Time-Space Network (TSN) models: in a TSN network, each airport is represented by a time line showing the planning horizon. Nodes show every departure/arrival at the corresponding airport time line and arcs show flights and connections (Hane et al., 1993). For example, in Figure 3.2, there are two airports, A and B . Solid arcs represent scheduled flights, while grey dashed arcs represent deadhead flights. A flight path between the two timelines, shown in blue, starts at airport A at time period 1 (node A_1), flies to airport B arriving at time period 2 (node B_2), then is grounded at airport B until time period 3 (node B_3), and so on;

Multi-Commodity Network Flow (MCNF) models: based on a fleet-flow time-space network (layered TSN models), each aircraft represents separate commodities and flow has to be preserved. Formulations of this type typically include constraints regarding capacities (passengers and fleet) and flow conservation (aircraft, flight, and airport) (Levin, 1971).

The first string-based formulations were introduced by Desaulniers et al. (1997),

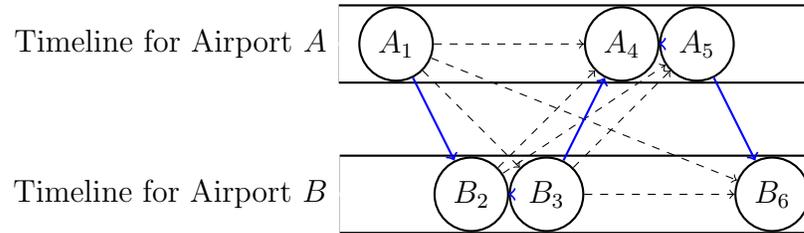


Figure 3.2: Time-Space Network (TSN) example.

for the FA problem, and Barnhart, Boland, Clarke, Johnson, Nemhauser and Shenoi (1998), for the TA problem. They both implemented a branch and bound scheme as their solution method. Even though they included maintenance regulations, workshop resources or FH were not considered. Sarac et al. (2006); Cohn and Barnhart (2003); Papadakos (2009) used the same type of formulation for the AMR problem. Papadakos (2009) produced a computational case study for a medium-sized data set (700 flights, 167 aircraft). The heuristic algorithm used required 16 hours to solve the problem under consideration. Sarac et al. (2006) implemented a branch-and-price algorithm and used legal remaining FH to influence decisions. Aside from the excessive solution times, string-based formulations are not capable of generating all strings even for small instances. Furthermore, these models do not include resource usage for maintenance activities.

The TSN formulation, introduced by Hane et al. (1993), has been used widely within the integrated airline scheduling framework, (Clarke et al., 1997; Hicks et al., 2005; Orhan et al., 2012; Haouari et al., 2013; Liang and Chaovalitwongse, 2013). However, given that TSNs do not allow individual aircraft to be tracked, aggregated maintenance constraints are implemented. This means that models are forced to, for instance, minimise total weekly maintenance operations. More recently, Safaei and

Jardine (2018) examined the AMR problem with generalised maintenance constraints and legal remaining FH considerations. They used test instances for a single airline. The computational study, for a relatively large data set (7 days, 772 flights, 18 aircraft), solutions show financial impact but computational times are not provided.

Using the MCNF formulation, introduced by Levin (1971), the problem can be solved using column generation (Yan and Tseng, 2002; Sriram and Haghani, 2003; Mercier et al., 2005). Particularly, Sriram and Haghani (2003), presented some influential work, based on the TA formulation by Feo and Bard (1989), which solved the AMR incorporating A and B checks. They used a heuristic algorithm that solved a small test instance (58 flights, 75 airports) in 5 minutes.

All the publications mentioned thus far are limited to daily or weekly schedules and assume cyclical repetitions of the flights and hence maintenance operations. Khaled et al. (2018), however, considered individual maintenance requirements for a 30-day plan. They used an improved TSN formulation for the AMR problem, which allowed them to effectively schedule A checks, constrained by individual aircraft legal remaining FH. They assumed that maintenance is generally performed at night. Due to the type of formulation, solution times increased noticeably for a large number of flights (timed out at 3 hours for 1494 flights and a single airline). Li et al. (2016), dealt with the AMR problem for fighter jets. This formulation, which relied on a single workshop assumption, also employed legal remaining FH to determine the frequency of maintenance. Further assumptions included: no resource considerations, only one type of maintenance, and the disregard of aircraft “health” at the end of the planning

horizon. The test set considered is of 25 days and 200 aircraft but no solution times are provided. In addition, the timeline is discretised by splitting each day into two 12 hour intervals, clearly, this incurs a huge loss in accuracy.

In different context, we find formulations that account for long-term planning horizons; specifically, in the resource constrained project scheduling problem (RCPSP) literature. The RCPSP is a generalisation of machine scheduling problem where jobs are scheduled according to some predefined order, or precedence, subject to different resource demands and capacity constraints. Usually, the objective is to minimise the duration of the project (collection of ordered jobs), commonly, under a non-preemption assumption (jobs have to be processed fully) (Koné et al., 2011; Brucker and Knust, 2012; Kopanos et al., 2014; Naber, 2017). The reader may refer to a recent and thorough review article by Habibi et al. (2018). On the long-term RCPSP, Koné et al. (2011) proposed two formulations which they named “event-based RCPSP” formulations. Events correspond to start and/or end times of activities. Since their formulations involve fewer variables than the formulations indexed by time, they have the capacity to deal with longer planning horizons.

As can be seen in the summary of the literature in Table 3.2, no publications have addressed the long-term AMS problem with multiple airlines, workshops and resource considerations, while providing fast solution times. Additionally, we take into account individual aircraft maintenance requirements and their respective flight operations. The key contributions of the present paper are,

Reassignment. We include the option of reassigning some flights to obtain longer

maintenance opportunities.

Efficient resource allocation. Maintenance models do not always consider the different resources available throughout maintenance workshops. Therefore, we incorporate workshop resource restrictions.

Individual aircraft considerations. Different aircraft may have different maintenance duration and requirements, and different accumulation of FH. Hence, we include this in our model.

Long-term planning horizon. Short-term, or operational planning, is not suitable for most aircraft maintenance. Further, personnel and equipment hire are significantly expensive. Thus, it is extremely useful for maintenance operators to plan longer in advance.

Two-stage iterative algorithm. We employ a two-stage algorithm that provides good solutions for large instances in reasonable computational time.

Single and multi-workshop tests cases. Our generic framework, allows us to model both single and multi-workshop cases.

Table 3.2: Summary of selected literature. MS = inclusion of maintenance scheduling, FH = use of legal remaining flying hours, RC = workshop resource considerations, H = time horizon (days), Sched. Type = scheduling type, Airlines = number of airlines considered, W = number of workshops considered, F = number of flights (in largest test instance), A = number of aircraft (in largest test instance), T = computational time (of largest test instance).

Article	MS	FH	RC	Sched. Type	Airlines	W	H	F	A	T
Desaulniers et al. (1997)	×	×	×	cyclical	single	1	1	383	91	1 h
Barnhart et al. (1998)	✓	×	×	cyclical	single	1	7	1124	89	10 h
Sriram and Haghani (2003)	✓	×	×	cyclical	single	20	7	rand.	13	4.5 h
Mercier et al. (2005)	✓	×	×	cyclical	two	NS	1	707	143	13 h
Sarac et al. (2006)	✓	✓	✓	cyclical	single	5	1	175	32	2 h
Papadakos (2009)	✓	×	×	cyclical	single	6	7	705	167	16 h
Haouari et al. (2013)	✓	×	×	cyclical	single	16	1	344	138	10 s
Liang and Chaovalitwongse (2013)	✓	×	×	cyclical	single	6	7	1780	110	4 h
Khaled et al. (2018)	✓	✓	×	adaptive	single	9	30	1494	40	3 h
Present paper	✓	✓	✓	adaptive	single	1	30	3869	49	1.2 h
Present paper	✓	✓	✓	adaptive	multiple	8	30	16000	529	1.8 h

3.3 The Proposed Modelling Approach

In order to be able to adequately model the problem, we first state the necessary assumptions, highlight the concepts used and define the appropriate notation. After this, we propose two multi-objective mixed integer linear programming (MMILP) formulations that schedule different types of maintenance for a medium/long-term planning horizon e.g. airframe checks A and C. To check if regulation requirements are being fulfilled, we employ different maintenance requirement (MR) variables for each type of maintenance and aircraft type, which vary with FH.

In the first formulation, given an input flight schedule, we aim to determine whether a feasible maintenance schedule exists. We call this the AMS formulation. If a feasible maintenance schedule does not exist, we seek to minimise the number of infeasibilities, or violations. Such violations represent the cases when the limit imposed by the regulations on the FH are exceeded. To minimise the violations, we introduce the second formulation which extends the AMS formulation to account for an appropriate TA problem.

3.3.1 Assumptions

To ease the formulation of the problem, we make some modelling assumptions.

1. Maintenance can only be performed in the pre-identified MOPs;
2. Maintenance can be done at any workshop, if the aircraft has a MOP there; unless otherwise specified by an airline;
3. Maintenance cannot be preempted;

4. To extend MOPs, we may reassign flights to different aircraft, i.e. we may modify the airlines' preferred TA;
5. Resources can be shared amongst different airlines at the maintenance workshops.

3.3.2 Concepts

The integrated airline scheduling literature reveals that the models used for short-term planning are not easily scalable for our 30-day planning horizon. Additionally, not much attention is paid to either resource usage for maintenance activities or aircraft health-state monitoring. However, as mentioned in the literature review section, Koné et al. (2011) presented a formulation for the long-term resource constrained project scheduling problem (RCPSP) that challenged the classical discretisation of time. Instead, they index the variables using some pre-defined “events”.

Koné et al. (2011), independently formalised the same idea as Sousa and Wolsey (1992), and refer to this type of formulation as an “event-based RCPSP”. Events represent either the start or the end of an activity. Compared to traditional time indexation, their formulations involve considerably fewer variables, therefore, being ideal for problems with long planning horizons. Koné et al. (2013) extended the formulation to account for non-renewable resources. More authors have used a similar indexation of continuous time. Naber (2017) focused on removing the assumption regarding fixed resources per activity in the RCPSP by allowing flexibility of resource usage.

In the present paper, events represent turnaround times, the time between arrival

and departure, at a certain maintenance workshop where the total time is sufficient to perform at least one type of maintenance. We regard these as MOPs. Thus, provided with a flight schedule, MOPs are easily identified and pose no restrictive assumption on the aircraft or airlines considered. For more information on how these are created, see Section 3.4.1.

3.3.3 Airline Fleet Maintenance Scheduling with Violations

The AMS with violations model checks, subject to regulations, whether a feasible maintenance schedule exists given a flight schedule. Violations of regulations, contrary to most other works, are not included as constraints but as an objective which we try to minimise. Hence, the impasse that infeasibilities bring is avoided and more conclusions can be inferred about the process. First, the necessary sets, parameters and variables are defined, then we introduce the MMILP formulation.

Notation

To formulate the problem, we introduce the following notation. The set of all aircraft is denoted by \mathcal{K} and indexed with k . In order to distinguish aircraft by type and airline, we introduce the set \mathcal{T} with function $t: \mathcal{K} \rightarrow \mathcal{T}$; $t(k)$ maps a specific aircraft $k \in \mathcal{K}$ to its corresponding fleet type and airline. Each aircraft has its own corresponding MOPs contained in the set MOP_k which we can index with j . Moreover, we can subdivide each MOP into time intervals. These provide an alternative discretisation of time and represent either the start or end time of some MOP. All time intervals are contained in the set \mathcal{I} , indexed by i . Each element i , has a start and end time, denoted with st^i and et^i respectively, and a maintenance workshop W^i . Some common

maintenance checks are collected in the set \mathcal{C} , indexed by c . Resources are contained in the set R . The demand for resource $r \in R$ varies per check $c \in \mathcal{C}$, this is denoted by b_{rc} . Moreover, the duration of maintenance may vary within aircraft types. Hence, for an aircraft k , of type $t(k)$, we define the duration of a single check $c \in \mathcal{C}$ to take $\Delta_{t(k),c}$ time units to complete.

For convenience, we can identify the following subsets. Let \mathcal{I}_k be the subset of intervals where aircraft k is available for maintenance. Similarly, let \mathcal{K}^i be the subset of aircraft available for maintenance at interval i . The resources available at the maintenance workshop of interval i , W^i , are contained in the set $R_{W^i} \subseteq R$.

Each MOP can be represented by a set of consecutive intervals. For $j \in MOP_k$, we can identify the corresponding intervals as $i \in MOP_k^j \subseteq \mathcal{I}_k$. This idea is illustrated in Figure 3.3, where an example for an aircraft k with three MOPs is shown. The numbering indicates the interval number. Each of MOP is labelled with a different $j = 1, 2, 3$. It is worth noting that every interval is assigned to exactly one MOP. We can identify the interval sets $\mathcal{I}_k = \{1, 2, 3, 7, 8, 11, 12\} \subset \mathcal{I} = \{1, \dots, 12\}$. Thus, given the definition of MOPs, intervals in $\mathcal{I} \setminus \mathcal{I}_k = \{4, 5, 6, 9, 10\}$ must belong to another aircraft (not pictured). Moreover, if i is part of a MOP for aircraft k , $i \in \mathcal{I}_k$, we can say that aircraft k is not flying at interval i , hence, is available for maintenance. Conversely, if i is not part of any MOP for aircraft k , $i \notin \mathcal{I}_k$, we can say that aircraft k is flying at interval i .

In order to employ FH accurately in the model, we introduce two additional parameters. Let FH_k^i be the FH for aircraft k at interval i , and, $R_{t(k),c}$ be the limit on

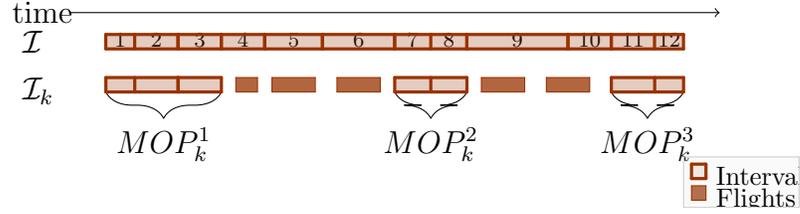


Figure 3.3: Timeline showing the deconstruction of three MOPs into sets of consecutive intervals.

FH imposed by regulations for aircraft of type $t(k)$ and check c . In order to update the FH parameter, we consider the flights between consecutive MOPs. For a given interval i and aircraft k , interval i is bounded by

$$\sup\{MOP_k^{j-1}\} < i \leq \sup\{MOP_k^j\}, \quad (3.3.1)$$

for some MOP_k^j provided that $j \neq 1$, $i > \sup\{MOP_k^1\}$ and $i \leq \sup\{MOP_k^J\}$ ($J = |MOP_k|$) (proof provided in A.2). That is, provided that the interval under consideration starts after the end of the first MOP and before the end of the last MOP. If this is the case, we define FH_k^i to consider only the flights that aircraft k operates between MOP_k^{j-1} and MOP_k^j . Otherwise, if i starts before the end of the first MOP, i.e. $i \leq \sup\{MOP_k^1\}$, we define FH_k^i to consider only the flights that aircraft k operates between the beginning of the planning horizon and MOP_k^1 . Conversely, if i starts after the end of the last MOP, i.e. $i > \sup\{MOP_k^J\}$, we define FH_k^i to consider only the flights that aircraft k operates between MOP_k^J and the end of the planning horizon. In Figure 3.3, if $i \in \{7, 8\} = MOP_k^2 \subset \mathcal{I}_k$, or $i \in \{4, 5, 6\} \notin \mathcal{I}_k$, we have $\sup\{MOP_k^1\} < i \leq \sup\{MOP_k^2\}$, then, FH_k^i considers the flights between MOP_k^1 and MOP_k^2 . If $i \in \{9, 10\} \notin \mathcal{I}_k$, we have $\sup\{MOP_k^2\} < i \leq \sup\{MOP_k^3\}$, then, FH_k^i considers the flights between MOP_k^2 and MOP_k^3 .

Using the previous two parameters, we can define the deterioration that a certain number of FH incurs on an aircraft for a specific check type. For an interval i , check c and aircraft k , the deterioration parameter is defined as,

$$DR_{kc}^i = \frac{FH_k^i}{R_{t(k),c}}. \quad (3.3.2)$$

In the next subsection, we provide the notation needed to describe the proposed mathematical model. However, it is useful to discuss two of the variables in detail. To account for the legal remaining FH, we define a variable, $w_{kc}^i \in [0, 1]$, that tracks the MR at the beginning of interval i , for check type c and aircraft k . Where a value close to 0 represents that no maintenance is required, and a value close to 1 represents that maintenance is urgently required. This variable is updated by using the deterioration parameter DR_{kc}^i . We define a binary variable to identify regulation violations, v_{kc}^i gets the value 1 if the regulation for check c is violated at interval i by aircraft k , while it is 0 otherwise.

Definitions

Sets

\mathcal{C} : Set of checks indexed by c ;

\mathcal{I} : Set of intervals indexed by i ;

\mathcal{I}_k : Set of intervals where aircraft k is available for maintenance, $\mathcal{I}_k \subseteq \mathcal{I}$;

\mathcal{K} : Set of all aircraft indexed by k ;

\mathcal{K}^i : Set of aircraft available for maintenance at interval i , $\mathcal{K}^i \subseteq \mathcal{K}$;

MOP_k : Set of MOPs for aircraft k , indexed by j ;

MOP_k^j : Set of intervals that constitute the j -th MOP for aircraft k , $MOP_k^j \subseteq \mathcal{I}_k$;

R : Set of resources indexed by r ;

R_{W^i} : Set of resources available at the workshop of interval i , W^i , $R_{W^i} \subseteq R$;

\mathcal{T} : Set of aircraft types with $t: \mathcal{K} \rightarrow \mathcal{T}$.

Parameters

b_{rc} : Demand of resource r to process check c ;

DR_{kc}^i : Deterioration (per flying hour) at interval i for check c and aircraft k ;

$\Delta_{t(k),c}$: Duration of check c for an aircraft of type $t(k)$;

st^i/et^i : Start/end time of interval $i \in \mathcal{I}$.

Variables

B_r^i : The capacity for resource r at interval i .

w_{kc}^i : A continuous variable with values between 0 and 1 to represent the MR for aircraft k for check c at the beginning of interval i . 0 means that the aircraft requires no maintenance, 1 means that the aircraft requires maintenance urgently.

m_{kc}^i : 1, if a maintenance check c for aircraft k starts/continues at the beginning of interval i ; 0, otherwise.

v_{kc}^i : 1, if regulation for check c is being violated at interval i for aircraft k ; 0, otherwise.

z_{kc}^i : 1, if there is a change between the consecutive variables, m_{kc}^i and m_{kc}^{i-1} ; 0, otherwise.

Formulation

Model 3.3.1. *Interval MMILP formulation for AMS with violations.*

$$\min \sum_k \sum_c \sum_i v_{kc}^i \quad (3.3.3)$$

$$\min \sum_k \sum_c w_{kc}^{\sup\{\mathcal{I}_k\}} \quad (3.3.4)$$

Subject to

Maintenance Requirement

$$v_{kc}^i = 0 \quad \forall k, c, i \notin \mathcal{I}_k; \quad (3.3.5)$$

$$m_{kc}^i = 0 \quad \forall k, c, i \notin \mathcal{I}_k; \quad (3.3.6)$$

$$w_{kc}^i \geq w_{kc}^{i-1} + DR_{kc}^i (1 - m_{kc}^i) - m_{kc}^i - v_{kc}^i \quad \forall k, c, i \neq \inf\{\mathcal{I}\}; \quad (3.3.7)$$

$$w_{kc}^{\sup\{MOP_k^j\}} \leq DR_{kc}^{\inf\{MOP_k^{j+1}\}} \quad \forall k, c, j \neq |MOP_k|; \quad (3.3.8)$$

Maintenance

$$\sum_c m_{kc}^i \leq 1 \quad \forall k, i \in \mathcal{I}_k; \quad (3.3.9)$$

$$\sum_{i' \in MOP_k^j} (et^{i'} - st^{i'}) m_{kc}^{i'} \geq \Delta_{t(k),c} m_{kc}^i \quad \forall k, c, j, i \in MOP_k^j; \quad (3.3.10)$$

Transitivity Constraints

$$z_{kc}^{\inf\{MOP_k^j\}} \geq m_{kc}^{\inf\{MOP_k^j\}} \quad \forall k, c, j; \quad (3.3.11)$$

$$z_{kc}^i \geq m_{kc}^i - m_{kc}^{i-1} \quad \forall k, c, j; i, i-1 \in MOP_k^j; \quad (3.3.12)$$

$$\sum_{i \in MOP_k^j} z_{kc}^i \leq 1 \quad \forall k, c, j; \quad (3.3.13)$$

Resources

$$\sum_{k \in \mathcal{K}^i} \sum_c b_{rc} m_{kc}^i \leq B_r^i \quad \forall i, r \in R_{Wi}; \quad (3.3.14)$$

Variables

$$w_{kc}^i \in [0, 1] \quad \forall k, c, i; \quad (3.3.15)$$

$$B_r^i \in \mathbb{R}^+ \quad \forall k, i, r \in R_{Wi}; \quad (3.3.16)$$

$$m_{kc}^i, z_{kc}^i, v_{kc}^i \in \{0, 1\} \quad \forall k, c, i. \quad (3.3.17)$$

The proposed formulation is a MMILP with two lexicographically ordered objec-

tive functions. The functions involved minimise the following objectives (in order of importance), the number of violations, and the total MR at the last interval. Objective 3.3.3 minimises the number of regulation violations. Objective 3.3.4 minimises the total MR at the end of the planning horizon. Recall that a value close to 0 for the w_{kc}^i variable indicates that the aircraft requires no maintenance; hence, minimising $w_{kc}^{\sup\{\mathcal{I}_k\}}$ corresponds to minimising the amount of maintenance required by each individual aircraft at the end of the planning horizon.

The first two MR constraints 3.3.5 and 3.3.6 ensure that neither a violation nor maintenance occur when an aircraft is operating flights. Precisely, constraints 3.3.5 and 3.3.6 ensure that if aircraft k is flying at interval i , i.e. $i \notin \mathcal{I}_k$, then, for any check c , neither a violation nor maintenance intervention may occur.

Constraints 3.3.7 enforce a recurrence relation for the MR variable. For time interval i , aircraft k , and check type c , we update the MR variable depending on whether or not aircraft k is flying at interval i . If aircraft k is not flying at interval i , i.e. $i \in \mathcal{I}_k$ and $i \in MOP_k^j$ for some j ; the current MR, w_{kc}^i , is updated using the previous MR, w_{kc}^{i-1} , plus the appropriate deterioration, DR_{kc}^i (the deterioration incurred by the flights between MOP_k^{j-1} and MOP_k^j , as defined in equation 3.3.2), or drops to 0 if either a violation or maintenance occur. Please recall that the first priority objective minimises the number of violations. On the other hand, if aircraft k is flying at interval i , i.e. $i \notin \mathcal{I}_k$, then, by constraints 3.3.5 and 3.3.6, $m_{kc}^i = 0$ and $v_{kc}^i = 0$, hence, the recurrence relation is enforced.

Constraints 3.3.8 ensure that the MR variable at the end of a MOP stays within the regulation limits (captured within the deterioration parameter) at least until the

next MOP. That is, for a given aircraft k , check type c and MOP j (with $j \neq |MOP_k|$), the MR at the end of the MOP, $w_{kc}^{\sup\{MOP_k^j\}}$, should remain feasible to operate the upcoming flights between MOP_k^j and MOP_k^{j+1} . Using the definition of the deterioration parameter, this is encapsulated in $DR_{kc}^{\inf\{MOP_k^{j+1}\}}$, since, by equation 3.3.1, $\sup\{MOP_k^j\} < \inf\{MOP_k^{j+1}\} \leq \sup\{MOP_k^{j+1}\}$.

Constraints 3.3.9 and 3.3.10 enforce the maintenance restrictions. Constraints 3.3.9 assure that no more than one maintenance type is scheduled for the same interval. Constraints 3.3.10 guarantee that the aircraft is available for the minimum time required for each maintenance type. More precisely, the sum of the duration of consecutive intervals has to be greater than the minimum prespecified duration of the check.

Transitivity constraints, 3.3.11, 3.3.12, and 3.3.13, ensure that if we decide to maintain in MOP_k^j , preemptions are not allowed (proof provided in A.2). Constraints 3.3.11 initialise the auxiliary variable using the first interval in the MOP. Constraints 3.3.12 establish that when a maintenance starts, i.e. the difference between consecutive maintenance variables is 1, the auxiliary variable is 1. Constraints 3.3.13 ensure that at most one auxiliary variable is 1, or equivalently we cannot start a maintenance more than once. Hence, wherever we terminate maintenance, all auxiliary variables thereafter must be 0.

Constraints 3.3.14 ensure that a maintenance intervention of some type is only scheduled if there are sufficient resources available at the workshop. The total number of checks over all aircraft present at a given interval cannot exceed the capacity for each resource. The last four constraints 3.3.15 – 3.3.17 define the domains of the

variables.

3.3.4 Airline Fleet Maintenance Scheduling with Tail Assignment

In this section we extend the AMS formulation previously discussed, to include reassignment variables for the periods where regulations are being violated. In order to determine where this occurs, we solve Model 3.3.1, and identify which violation variables, v_{kc}^i , have the value 1. Using this information we efficiently select reassignable and preassigned flights which allow us to solve the joint AMS and TA problem.

Notation

To formulate the AMS with TA, we first expand the notation introduced for Model 3.3.1. All flights are contained in the set \mathcal{F} , which we can index with f . More precisely, it contains flight legs (sequence of multiple flights) between MOPs. Also, let \mathcal{I}^f be the subset of intervals at which flight $f \in \mathcal{F}$ occurs, and \mathcal{K}^f be the subset of aircraft free to operate flight f .

Additionally, for a given interval i , we can identify the set of flights that either, depart at the end of interval i , or, arrive at the start of interval i . Let us denote the departure set with

$$\mathcal{F}_{dep}^i = \{f : f \in \mathcal{F}, \inf\{\mathcal{I}_f\} = i\} ,$$

and, the arrival set with

$$\mathcal{F}_{arr}^i = \{f : f \in \mathcal{F}, \sup\{\mathcal{I}_f\} = i\} .$$

In order to control the number of reassignment variables, we can identify the subsets of flights which are *reassignable* and those which are fixed or *preassigned*. Such that, if a flight is reassignable, then we can reassign it to another aircraft; in contrast, if a flight is preassigned, then it is operated by the preassigned aircraft. Let $\mathcal{F}_R \subset \mathcal{F}$ and $\overline{\mathcal{F}}_R \subset \mathcal{F}$ be two (disjoint) subsets (with $\mathcal{F}_R \cup \overline{\mathcal{F}}_R = \mathcal{F}$), that contain those flights which are reassignable and preassigned, respectively. Also, let O^f for flight $f \in \overline{\mathcal{F}}_R$, with $O^f \subseteq \mathcal{K}^f$, contain the aircraft preassigned to flight f . To preserve efficiency and minimise the changes in the airlines' preferred TA, instead of setting all flights to be reassignable, we select an appropriate subset of flights. More details on the selection process can be found in Section 3.4.2.

We can redefine the deterioration parameter in terms of flights, DR_{kc}^f , which represents the deterioration for the operation of flight f for check c and aircraft k . Similarly, as for the previous model,

$$DR_{kc}^f = \frac{FH^f}{R_{t(k),c}},$$

where FH^f are the FH that correspond to flight f . Lastly, the violation variable can also be expressed in terms of flights, v_{kc}^f gets the value 1 if a regulation is being violated before flight f for aircraft k and check c , while it is 0 otherwise.

Definitions

Sets

\mathcal{F} : Set of all flights indexed by f ;

\mathcal{F}_R : Set of reassignable flights, $\mathcal{F}_R \subset \mathcal{F}$;

$\overline{\mathcal{F}}_R$: Set of preassigned flights, $\overline{\mathcal{F}}_R \subset \mathcal{F}$;

\mathcal{F}_{dep}^i : Set of flights which are scheduled to depart at the end of interval i ;

\mathcal{F}_{arr}^i : Set of flights which are scheduled to arrive at the start of interval i ;

\mathcal{I}^f : Set of intervals occupied by flight f , $\mathcal{I}^f \subseteq \mathcal{I}$;

\mathcal{K}^f : Set of aircraft available to operate flight f , $\mathcal{K}^f \subseteq \mathcal{K}$;

O^f : Set of aircraft preassigned to operate flight $f \in \overline{\mathcal{F}}_R$, $O^f \subseteq \mathcal{K}^f$.

Parameters

DR_{kc}^f : Deterioration for the operation of flight f for check c and aircraft k .

Variables

a_k^f : 1, if the flight f is (re)assigned to aircraft k ; 0, otherwise.

\overline{B}_r : The maximum capacity for resource r .

v_{kc}^f : 1, if the regulation for check c is being violated before flight f for aircraft k ; 0, otherwise.

Formulation**Model 3.3.2.** *Interval MMILP formulation for AMS with TA.*

$$\min \sum_f \sum_{k \in \mathcal{K}^f} \sum_c v_{kc}^f \quad (3.3.18)$$

$$\min \sum_r \bar{B}_r \quad (3.3.19)$$

$$\min \sum_f \sum_{k \in \mathcal{K}^f \setminus O^f} a_k^f \quad (3.3.20)$$

$$\min \sum_k \sum_c \sum_i (et^i - st^i) m_{kc}^i \quad (3.3.21)$$

$$\min \sum_i \sum_r (et^i - st^i) B_r^i \quad (3.3.22)$$

$$\min \sum_k \sum_c w_{kc}^{\sup\{\mathcal{I}_k\}} \quad (3.3.23)$$

Subject to*Preassigned Flights*

$$a_k^f = 1 \quad \forall f \in \overline{\mathcal{F}}_R, k \in O^f; \quad (3.3.24)$$

$$\sum_c v_{kc}^f = 0 \quad \forall f \in \overline{\mathcal{F}}_R, k \in O^f; \quad (3.3.25)$$

Maintenance Requirement

$$w_{kc}^i \geq w_{kc}^{i-1} + \sum_{f \in \mathcal{F}_{arr}^i} (DR_{kc}^f a_k^f - v_{kc}^f) - m_{kc}^i \quad \forall k, c, i \neq \inf\{\mathcal{I}\}; \quad (3.3.26)$$

$$w_{kc}^{\sup\{MOP_k^j\}} \leq \sum_{f \in \mathcal{F}_{dep}^{\sup\{MOP_k^j\}}} DR_{kc}^f a_k^f \quad \forall k, c, j; \quad (3.3.27)$$

Reassignment

$$m_{kc}^{i+1} \leq 1 - \sum_{f \in \mathcal{F}_{dep}^i} a_k^f \quad \forall k, c, i \neq \sup\{\mathcal{I}\}; \quad (3.3.28)$$

$$\sum_k a_k^f = 1 \quad \forall f; \quad (3.3.29)$$

*Maintenance**Constraints 3.3.6, 3.3.9 and 3.3.10**Transitivity Constraints**Constraints 3.3.11 – 3.3.13**Resources**Constraints 3.3.14*

$$B_r^i \leq \bar{B}_r \quad \forall i, r \in R_{Wi}; \quad (3.3.30)$$

$$(3.3.31)$$

Variables

$$a_k^f \in \{0, 1\} \quad \forall k, f; \quad (3.3.32)$$

$$v_{kc}^f \in \{0, 1\} \quad \forall k, c, f; \quad (3.3.33)$$

$$\overline{B}_r \in \mathbb{R}^+ \quad \forall r \quad (3.3.34)$$

Constraints 3.3.15 – 3.3.17

The proposed formulation is a MMILP with six lexicographically ordered objective functions. The functions involved minimise the following objectives (in the order of importance), the number of violations, maximum resource level, number of reassigned flights, number of maintenance interventions, overall resource usage, and total MR. Objective 3.3.18, with largest priority, minimises the number of regulation violations. Objective 3.3.19 minimises the sum of maximum level for each resource. Objective 3.3.20 minimises the number of strictly reassigned flights (i.e. assigned to an aircraft different from the one in the input TA), thus minimising the number of changes in the airlines' preferred TA. Objective 3.3.21 minimises the number of maintenance interventions weighted with the duration of intervals. Objective 3.3.22 minimises the resource level per interval, again, weighted with the duration of intervals. Finally, as in the previous model, with least priority, objective 3.3.23, minimises the total MR at the last interval.

Constraints 3.3.24 ensure that if flight f is preassigned to aircraft $k \in O^f$, then $a_k^f = 1$. Similarly, constraints 3.3.25 ensure that if flight f is preassigned to aircraft $k \in O^f$, then a violation cannot occur $v_{kc}^f = 0$ for any check c .

The MR constraints are a simple extension of those in Model 3.3.1. Constraints 3.3.26 enforce a recurrence relation where the current MR, w_{kc}^i , is updated using the

previous MR, w_{kc}^{i-1} , plus a deterioration term if the aircraft has been assigned the flights prior to the interval under consideration, or drops to 0 if either a violation or maintenance occurs. In the case when flight f is reassignable, i.e. $f \in \mathcal{F}_R$, and if flight f is not reassigned to aircraft k , $a_k^f = 0$, then the MR is only updated if a violation or maintenance occurs. On the other hand, in the case of either, flight f being reassigned to aircraft k , or, flight f being preassigned to aircraft k ($f \in \overline{\mathcal{F}_R}$ and $k \in O^f$), we have, $a_k^f = 1$. Thus, by constraints 3.3.25 and 3.3.28, $v_{kc}^f = 0$ and $m_{kc}^i = 0$, thus, the MR is deteriorated according to the appropriate deterioration, DR_{kc}^f . Therefore, the recurrence relation is enforced.

Constraints 3.3.27 ensure that the MR remains feasible for the operation of any of the flights that depart the end of interval i . That is, for a given aircraft k , check type c and MOP j , the MR at the end of the MOP, $w_{kc}^{\sup\{MOP_k^j\}}$, should remain feasible to operate whichever flights are assigned to aircraft k departing at the end of the MOP. Such flights are contained in $\mathcal{F}_{dep}^{\sup\{MOP_k^j\}}$.

As for the reassignment, constraints 3.3.28 ensures that maintenance is not performed if an aircraft departs. More specifically, if an aircraft k is reassigned and due to depart on a flight after interval $i \neq \sup\{\mathcal{I}\}$, then, at interval $i+1$ (after the aircraft has departed) maintenance cannot be performed, hence, $m_{kc}^{i+1} = 0$. Constraints 3.3.29 ensure that all flights have exactly one aircraft assigned to them.

Constraints 3.3.30 establish the value for the maximum resource level for each resource. The remaining constraints, (maintenance, transitivity and resource constraints) as well as variable definitions, can be borrowed from Model 3.3.1. The last constraints 3.3.32–3.3.34, define the domains of the extra variables.

3.4 Solution Methodology

The solution approach chosen only requires flight schedules, and resource capacities and demands for maintenance services. To improve the efficiency of the solutions, after a preprocessing routine, we implement an iterative algorithm. The algorithm is displayed in Figure 3.4 and pseudocode presented in Algorithm 3. The algorithm consists of two stages, *conflicting period selection* and *interval splitting*. The conflicting period selection stage involves selecting the sets of reassignable and preassigned flights, and resolving. This stage terminates when all the violations are removed or when the size of the conflicting periods cannot be increased any further. During subsequent iterations, the interval splitting stage identifies intervals where maintenance occurs, splits them and resolves the problem. Splitting time intervals allows the model to assign more maintenance to the existing schedule since it makes time intervals more granular.

3.4.1 Preprocessing Routine

Flight schedules are crucial for the model as they give the initial TA and accurate FH in order to update the MR throughout the planning horizon. We obtained flight schedules from Flightradar24 AB (2018) using `pyflightdata`, the Python module (Allamraju, 2014). We gathered data globally for an extended period. Following the data gathering stage, we preprocess the flight schedule data. Preprocessing involves filtering schedules through the nearest airport to the maintenance workshops under consideration. After this, we identify airlines and aircraft types of interest so we can track and update the aircraft's FH appropriately. This gives a reduced network

with accurate FH for each aircraft. For eight maintenance workshops over a 30-day planning horizon (between dates 14/11/16 and 15/12/16), prior to preprocessing, we have 23927 flights and 1643 aircraft of two types (Airbus A320 and Fokker 100). Then, we proceed to identify MOPs i.e. turnaround times sufficient to perform at least the shortest maintenance type. We choose a turnaround time of at least 5 hours to allow for at least a short maintenance intervention. Using aircraft MOPs, we generate intervals by identifying all start and end times of the MOPs and storing them in an ordered set.

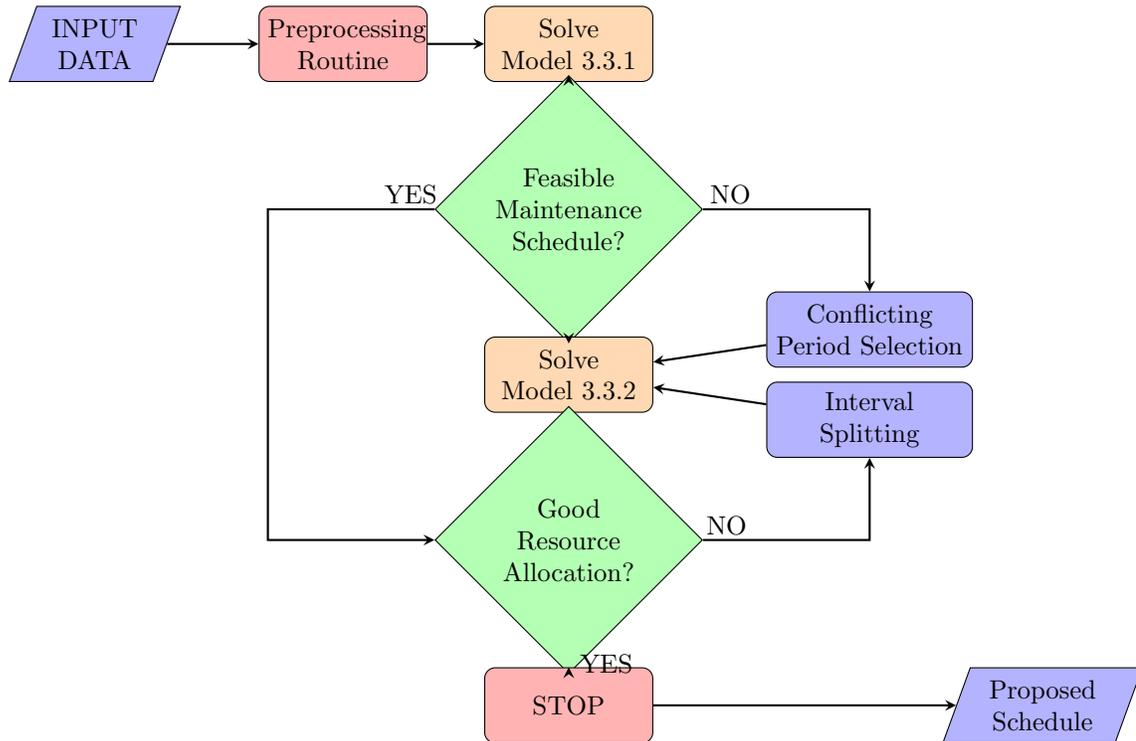


Figure 3.4: Flow chart outlining the process of the iterative algorithm.

As part of this stage, we pre-determine which variables are not required in the model, hence, for efficiency, they are not created. Specifically, all the variables involved in constraints 3.3.5 and 3.3.6 in Model 3.3.1, and constraints 3.3.24, 3.3.25, and 3.3.28

in Model 3.3.2, have known values and it is unnecessary to create them. Therefore, in practice, since the reassignment variables that involve preassigned flights are not present in Model 3.3.2, when MR constraints involve preassigned flights, we simply use MR constraints from Model 3.3.1. For this reason, the number of reassignment variables is determined by selecting the sets of reassignable and preassigned flights.

3.4.2 Solution Procedure

To ensure efficiency and solution accuracy, we propose an iterative solution procedure which has two stages, conflicting period selection, and interval splitting. The aim of the conflicting period selection stage is to, by using the identification of regulation violations, select the number of reassignment variables. In each iteration, if the regulation violations involved have not been removed, we increase the size of the conflicting period (which determines the number of reassignment variables) and resolve Model 3.3.2. In the interval splitting stage, once all infeasibilities have been removed or all reassignment variables have been introduced, we split intervals where maintenance takes place and resolve Model 3.3.2. This ensures flexibility as it allows the additional generated intervals to be allocated to different aircraft.

Conflicting Period Selection

An illustration of the conflicting period selection stage is given in Figure 3.5. For a certain violation, say at interval i , with $\{k_1, k_2, k_3, \dots, k_K\} \in \mathcal{K}^i$, we select the initial conflicting period as shown in red with $j_{(1)}^i$ as the upper bound, as the next interval where $\mathcal{K}^i \subseteq \mathcal{K}^{j_{(1)}^i}$. We classify flights either in the reassignable subset (\mathcal{F}_R), if they are involved in the conflict, or in the preassigned subset ($\overline{\mathcal{F}_R}$), otherwise. This

regulates the number of reassignment variables, and keeps the solution process of Model 3.3.2 efficient. If in the new solution, interval i still has a violation, we increase the size of the conflicting period to reach $j_{(2)}^i$, the next interval where $\mathcal{K}^i \subseteq \mathcal{K}^{j_{(2)}^i}$. Again, we update the sets of reassignable and preassigned flights and resolve the problem. If the violation has not been removed, we move on to the next interval with matching aircraft. We continue this process until the violation is removed or the end of the planning horizon is reached, in which case, all the reassignment variables, corresponding to violation i , would have been introduced.

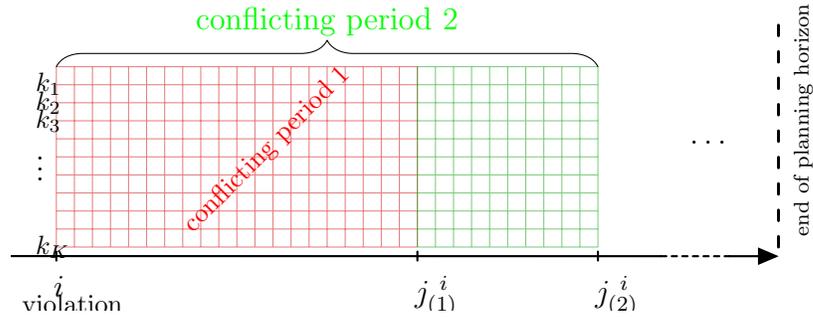


Figure 3.5: Conflicting period selection.

In order to identify the conflicting period, we have to identify where violations occur. For the first iteration, the set V that contain all intervals for which at least one violation occurs is given by,

$$V = \left\{ i : i \in \mathcal{I}, \sum_k \sum_c v_{kc}^i \geq 1 \right\},$$

thereafter, it is given by,

$$V = \{ i : i \in \mathcal{I}^f, f \in F \},$$

where,

$$F = \left\{ f : f \in \mathcal{F}, \sum_k \sum_c v_{kc}^f \geq 1 \right\} .$$

For every interval with a violation, $i \in V$, we can identify the set of aircraft involved, \mathcal{K}^i . Then, we can find the interval where the aircraft in \mathcal{K}^i will meet again, say \mathcal{K}^j , with $\mathcal{K}^i \subseteq \mathcal{K}^j$, where $j > i$. Hence, the reassignable flights, \mathcal{F}_R , are those that occur between intervals i and j , the conflicting period, and for $k \in \mathcal{K}^i$. All the flights, outside of the conflicting period are preassigned, thus, $\overline{\mathcal{F}_R} = \mathcal{F} \setminus \mathcal{F}_R$. More generally, the set of intervals J^i , which contain the set of aircraft \mathcal{K}^i , can be written as,

$$J^i = \{j : j \in \mathcal{I}, j > i, \mathcal{K}^i \subseteq \mathcal{K}^j\} . \quad (3.4.1)$$

With this, we can write the first conflicting period as

$$CP(1, i) = \{i' : i' \in \mathcal{I}, i \leq i' \leq j_{(1)}^i, \exists k \in \mathcal{K}^{i'} \wedge k \in \mathcal{K}^i\} , \quad (3.4.2)$$

where $j_{(1)}^i$ is the first element in J^i . We use this to update the sets of reassignable and preassigned flights appropriately. The set of reassignable flights is given by,

$$\mathcal{F}_R = \{f : \inf\{\mathcal{I}^f\} \geq i, \sup\{\mathcal{I}^f\} \leq j_{(1)}^i\} ,$$

and $\mathcal{K}^f = \mathcal{K}^i$ for $f \in \mathcal{F}_R$. The set of preassigned flights can be found using the updated set of reassignable flights, while O^f , the set of aircraft preassigned to operate flight $f \in \overline{\mathcal{F}_R}$ is given by the initial TA. Additionally, we need to update the aircraft present at the intervals in the conflicting period. Thus, we set

$$\mathcal{K}^{i'} = \mathcal{K}^{i'} \cup \mathcal{K}^i \text{ for } i' \in CP(1, i) .$$

By solving Model 3.3.2 after these updates, we can reassign the flights to any of the aircraft involved in the conflict. In the next iterations, if the violation is removed, then without loss of generality, we may assume that the remaining schedule can remain unchanged. However, if solving Model 3.3.2 has not led to the elimination of the violation, we expand the conflicting period and resolve. For this, we use $CP(m, i)$, for $m = 2, \dots, |J^i|$, which uses the m -th element of J^i , $j_{(m)}^i$. Similarly, to update the set of reassignable flights, we set,

$$\mathcal{F}_R = \{f : \inf\{\mathcal{I}^f\} \geq i, \sup\{\mathcal{I}^f\} \leq j_{(m)}^i\} ,$$

with, $\mathcal{K}^f = \mathcal{K}^i$ for $f \in \mathcal{F}_R$, and,

$$\mathcal{K}^{i'} = \mathcal{K}^{i'} \cup \mathcal{K}^i \text{ for } i' \in CP(m, i) .$$

With the set of preassigned flights and aircraft being updated as previously, using the new set of reassignable flights. We do this for every interval with a violation $i \in V$, until either the violation is eliminated or the end of the planning horizon is reached.

Interval Splitting

The interval splitting stage favours the redistribution of resources by using a more granular timeline with each iteration. Resources are occupied for the duration of the interval if maintenance is being performed. When long intervals occur, given the non-preemption assumption, resources can be held even after the maintenance has been

finalised (exceeding the minimum maintenance duration). Therefore, we consider the effect of splitting intervals, using different criteria, and resolving the problem. For example, in Figure 3.6, given the intervals, in the first iteration we have a MOP of 9 hours. Suppose that the maintenance scheduled for this MOP only takes 8 hours. So, in the first iteration, the last hour of the last interval is being wastefully allocated. Splitting, therefore, allows for the resources to be allocated to different aircraft. By splitting in half, in iteration 3, we see that the last hour is no longer being held.

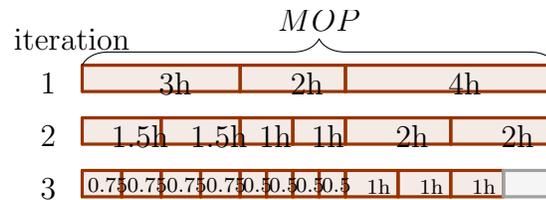


Figure 3.6: Interval splitting stage for a 9 hour MOP using binary segmentation for three iterations.

Let us define three simple splitting methods. Suppose, after solving the problem, we have $m_{kc}^i = 1$, so we split interval i using the following,

1. Binary segmentation (split in half);
2. Golden ratio (split by Golden ratio);
3. Minimum cut (split that allows at least the shortest type of maintenance).

Methods 1 and 2 are well-known and are regularly employed in search algorithms (Nocedal and Wright, 2006). Method 3 splits intervals that allows the shortest type of maintenance. Going back to the example in Figure 3.6, recall that the duration of the maintenance is 8 hours, splitting using Method 3 would produce a single split at the 8th hour in the second iteration.

To compare the splitting methods and identify good solutions, we compute an accuracy measure representing the usage of MOPs. The accuracy measure can be expressed as the ratio of the minimum time required for a check over the actual time scheduled for the check. That is,

$$A = \frac{\sum_c \sum_k \sum_j \Delta_{t(k),c}}{\sum_c \sum_k \sum_j \sum_{i \in MOP_k^j} (et^i - st^i) m_{kc}^i} . \quad (3.4.3)$$

Given constraints 3.3.10, which specify that maintenance scheduled should be at least of the minimum required duration; we have that $A \leq 1$. Therefore, a schedule that has an accuracy value close to 1, is one that does not schedule more maintenance than strictly required, and is, thus, efficient.

Algorithm

As outlined in Figure 3.4 and in the pseudocode of Algorithm 3, the algorithm requires some inputs. Specifically, a set of intervals, set of aircraft, FH for each aircraft, maintenance regulations and durations, and, resource capacities and demands. Once these are provided, in the first iteration, Model 3.3.1 is solved. If there are violations; i.e. there is no feasible maintenance schedule or, equivalently, $V \neq \emptyset$; we implement the conflicting period selection stage and resolve Model 3.3.2 until the violations have been removed. After this, we iterate the interval splitting stage and resolve Model 3.3.2 until either, the resource allocation is good enough, equivalently, $A \geq 1 - \epsilon$ (where ϵ is the tolerance), or, no more intervals are added.

Algorithm 3 Solution procedure with conflicting period selection and interval splitting.

```

1: Initialisations iter,  $A = 0$            ▷ Iteration counter and accuracy measure
2:  $\epsilon = 0.01$                                ▷ Set Tolerance
3: maxiter = 50                                ▷ Maximum number of iterations
4: Intervals =  $\mathcal{I}$                           ▷ Interval set
5:  $V, CP, oldIntervals = []$ 
6: while iter < maxiter and  $A < 1 - \epsilon$  do
7:   if iter = 0 then
8:     Solve Model 3.3.1                          ▷ For the first iteration
9:   else
10:    Solve Model 3.3.2                          ▷ For iterations  $\geq 1$ 
11:   end if
12:   Update  $V$  and  $A$ 
13:   if  $V$  is not empty then                   ▷ If violations have not been removed
14:     SELECTCP( $V$ )                             ▷ Call function to update conflicting periods
15:   else
16:     which =  $\{i: i \in \mathcal{I}, \sum_k \sum_c m_{kc}^i \geq 1\}$ 
17:     oldIntervals = Intervals
18:     SPLITINTERVALS(which, Intervals)      ▷ Call function to split intervals
19:     if  $|\mathbf{Intervals}| = |\mathbf{oldIntervals}|$  then  ▷ No intervals have been added
20:       Break                                  ▷ Stop the algorithm
21:     end if
22:   end if
23:   Increment iter
24: end while
25: function SELECTCP( $V$ )                       ▷ Updates the conflicting periods for all violations
26:   for  $i$  in  $V$  do
27:     Compute  $J^i$ 
28:     if  $i$  not in  $CP$  then                       ▷ If the violation is new
29:        $CP = CP \cup CP(1, i)$                    ▷ Update  $CP$  with first conflicting period
30:     else                                       ▷ If violation has occurred at a previous iteration
31:       for  $m$  in  $2 \rightarrow |J^i|$  do           ▷ Iterate through elements in  $J_i$ 
32:         if  $CP(m-1, i)$  in  $CP$  then           ▷ Find already used  $CP$ 
33:            $CP = CP \setminus CP(m-1, i)$        ▷ Remove it from conflicting periods
34:            $CP = CP \cup CP(m, i)$ 
35:         Break
36:       end if
37:     end for
38:   end if
39: end for
40:    $\mathbf{Intervals} = \mathbf{Intervals} \cup \mathbf{CP}$        ▷ Update intervals
41: end function
42: function SPLITINTERVALS(which, Intervals)  ▷ Splits intervals in which
43:   for  $i$  in which do
44:     Using Method 1, 2, or 3; split  $i$  into  $i_1$  and  $i_2$ 
45:      $\mathbf{Intervals} = \mathbf{Intervals} \setminus \{i\} \cup \{i_1, i_2\}$   ▷ Update Intervals
46:   end for
47: end function

```

3.5 Model Application and Computational Tests

We tested the iterative algorithm using flight schedules obtained for the maintenance workshops under consideration over the 30-day period selected. The results shown in this section use preprocessed flight data between dates 14/11/16 and 15/12/16 (see Section 3.4.1 for more information on the data gathering and preprocessing). The iterative algorithm was written in Python, using Gurobi Optimization version 8.1 (2019) to solve the models. Two computational studies are presented, one for the single workshop case (five workshops treated independently) and one for the multi-workshop case (up to eight workshops treated simultaneously).

As for the experimental set-up, we restrict the interval splitting stage of the algorithm such that the resulting intervals are at least 5 seconds long. Solution times for each iteration are limited to 500 seconds for each objective. The parameters used for the computational tests include the standard duration and frequencies for the regulated medium-term maintenance checks, as mentioned in Section 3.1. Specifically, according to Table 3.1, for the two types of aircraft under consideration, the values of regulation parameter, $R_{t(k),c}$, is given in Table 3.3. Here, we see the type of aircraft (Airbus A320 and Fokker 100), check type ($c = 1$ or 2) and the corresponding maintenance regulation parameter values, $R_{t(k),c}$. Using these values, $c = 1$ corresponds to an A check and $c = 2$ corresponds to a C check. Resource demand and capacity bounds, shown in Table 3.4, vary for four different types of renewable resources. A realistic interpretation of resources is as follows, r_1 – number of hangar bays, r_2 – certified technicians, r_3, r_4 – different types of specialised tools. The value for the

tolerance, ϵ , which determines the required accuracy level as $A \geq 1 - \epsilon$, is set to be 0.01. The initial MR variable is sampled from a Uniform distribution as follows,

$$w_{kc}^{\inf\{\mathcal{I}_k\}} \sim Unif(0, 0.3) \quad \forall k, c.$$

Sensitivity analysis around the chosen value produces distinct maintenance schedules and resource profiles but does not affect the computational performance of the algorithm. For instance, higher values lead to more maintenance being scheduled towards the start of the planning horizon; conversely, lower values lead to more maintenance being scheduled towards the end of the planning horizon.

Table 3.3: Maintenance regulation parameter values for two check types (1 and 2) and two aircraft types (A320 and F100).

$t(k)$	$c = 1$	$c = 2$
Airbus A320	$R_{t(k),c} = 600\text{FH}$	$R_{t(k),c} = 6000\text{FH}$
Fokker 100	$R_{t(k),c} = 500\text{FH}$	$R_{t(k),c} = 5000\text{FH}$

Table 3.4: Sample resource demands and limit capacities for four types of resources (r_i for $i = 1, 2, 3, 4$) and two maintenance types (1 and 2).

r	b_{r1}	b_{r2}	\bar{B}_r
r_1	1	1	25
r_2	3	5	25
r_3	2	3	25
r_4	1	2	25

In addition, the formulations are solved using lexicographic ordered objectives with priorities as suggested by airline practitioners. In the order of importance, the objectives which are minimised, namely,

1. the number of violations (objective 3.3.18),

2. the maximum resource level (objective 3.3.19),
3. the number of strictly reassigned flights (objective 3.3.20),
4. the weighted number of maintenance interventions (objective 3.3.21),
5. the total MR at the end of the planning horizon (objective 3.3.23).

Given that we are seeking for a maintenance feasible schedule, assigning the number of violations, in any but the highest priority level leads to infeasible maintenance schedules. Apart from this, as tests revealed, the order of the other objectives does not affect the integrity of the solutions or the computational times.

3.5.1 Single Workshop Case

In the case when maintenance workshops can be treated independently, we can solve the problem for each workshop individually. This situation can occur, for example, due to the geographical location of the workshops or upon a particular airlines' request or restrictions. In this case, we implement the algorithm in parallel for each maintenance workshop under consideration. It is worth noting that due to the preprocessing routine, which leads to considering only flights with large turnaround times and just two specific aircraft types, the number of flights and aircraft are significantly reduced compared those seen in ordinary operations; for more information see Section 3.4.1. The details for the five workshops, are as follows,

Atlanta Hartsfield-Jackson International Airport: with 1048 flights and 115 aircraft, produces 389 intervals;

Bangkok Suvarnabhumi Airport: with 3869 flights and 49 aircraft, produces 843 intervals;

Cairo International Airport: with 781 flights and 34 aircraft, produces 279 intervals;

Dubai International Airport: with 223 flights and 11 aircraft, produces 63 intervals;

Tokyo Haneda International Airport: with 978 flights and 10 aircraft, produces 162 intervals.

In all cases under consideration, after a single iteration of the conflicting period selection stage, violations are removed and the interval splitting stage begins. In order to compare the three splitting methods, we study four different aspects, namely, the final number of intervals, total run times, accuracy measure, and objective function value. Table 3.5 breaks down the objective function for the largest workshop, while the results for the workshops under consideration are shown in Table 3.6. Specifically, Table 3.6 shows the accuracy measure plots per iteration, final number of intervals, and total run times. In the accuracy measure plots, as shown in the legend, Method 1 is represented with solid lines, Method 2 with dashed lines, and Method 3 with dot-dashed lines.

The trend with the number of intervals per iteration is increasing for all methods, which is expected. The final number of intervals, as shown in Table 3.6, is the number of intervals at the last iteration. Method 3 offers the least final number of intervals

throughout, which is reflected in its solution times. Method 1 reveals a higher number of intervals than Method 2.

The run times for the largest workshop, Bangkok, using Method 3 takes 72 minutes to terminate the algorithm. For the remaining, small to medium-sized workshops, it takes between 0.36 seconds to 4 minutes to reach a good solution. Computational times for Method 1 and Method 2 (both significantly larger than Method 3) with the latter showing lower solution times.

The accuracy measure per iteration appears as a plot in the second column in Table 3.6. As can be seen, the accuracy measure evolves differently depending on the workshop. It takes a varying number of iterations across workshops for the algorithm to terminate. In all cases, more clear for Atlanta and Dubai, the required level of accuracy (0.99) is not reached; the algorithm terminates due to no more intervals being created. For the cases of Tokyo and Dubai, using Method 3, terminates the algorithm in very few iterations; whereas the rest take slightly more. Between Methods 1 and 2, both provide very similar quality solutions.

A breakdown of the different components of the objective function value for the Bangkok workshop is given in Table 3.5. The table shows the different values for each objective per iteration and for the corresponding splitting method. In each iteration, the best method is shown in green. This is determined by comparing objectives in decreasing order of priority until differing objectives are found, and one method presents an objective value lower than the rest. Note that objectives 3.3.18 (number of violations) and 3.3.20 (number of reassigned flights) remain constant. Objectives 3.3.21 (weighted number of maintenance interventions) and 3.3.22 (weighted overall

resource usage) decrease, while objectives 3.3.19 (maximum resource level) and 3.3.23 (total MR at the end of the planning horizon) show some fluctuation. The decrease in the fourth and fifth priority objectives is due to them being the only ones weighted with the duration of the intervals, and, therefore, are the only ones that are decreasing as the duration of maintenance interventions also decrease. The first iteration is the starting point where no splitting has occurred, hence, objectives have the same values throughout the three splitting methods. In the second iteration, Method 2 is better than Method 3 as it presents the same first three priority objectives (Obj. 3.3.18 - 3.3.20) and a lower value in the fourth priority objective (Obj. 3.3.21). Iterations thereafter show that Method 3 dominates with a lower second priority objective (Obj. 3.3.19). Additionally, Method 3 provides healthier fleet overall, as suggested by the consistently lower value of objective 3.3.23.

Due to its good solution times, lowest objective values, and overall higher accuracy measure we can claim that, for the single workshop case, the interval splitting stage performs better using Method 3. Furthermore, it is worth noting that the aircraft are, also, kept in a healthier state at the end of the planning horizon. For this reason, we present more detailed results using Method 3 for the largest workshop (Bangkok).

Figure 3.7 shows two resource profiles (first and last iteration) for resource r_1 at the Bangkok workshop. From Figure 3.7a to Figure 3.7b, we see that the resource profiles become considerably less populated and thinner. This means that the resource usage is more efficient. Particularly, by the last iteration, the solution tends to have slightly higher resource levels to avoid performing maintenance during busy flight periods.

This leads to most maintenance occurring during night and early morning shifts, as one would expect.

Since terminating the algorithm when $A \geq 1 - \epsilon$, or when no more intervals can be added, does not guarantee an optimal solution we conducted some further testing. For the Atlanta workshop, we compare our solution (using Method 3) to one obtained with a traditional discretisation method. We discretised time intervals using a varying time step, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15 minutes, and solved the problem once with a 5 hour time restriction per objective. Figure 3.8 shows, on the left-hand y -axis, the percentage error when comparing the optimal objectives 3.3.21 (weighted number of maintenance interventions) obtained using our method with the traditional discretisation for different time steps. The right-hand y -axis, with a grey dashed line, shows the computational times when using the traditional discretisation for different step sizes.

The percentage error for objectives 3.3.21, and 3.3.22, are shown using a solid red, and blue lines respectively. The corresponding error for the lower bounds are shown in the same colours but with dot-dashed lines. As with previous cases, objectives 3.3.18–3.3.20 remain constant, so they have not been included in the figure. It can be observed that the percentage error increases as the time step decreases. This means that the solution obtained with the traditional discretisation is improving, with respect to our optimal solution, as the step size is reduced. Our solution provides better solutions for step sizes of 8 minutes and above (hence the negative percentage error values). The optimal solution for a time step of 4 minutes (the smallest available as 2 and

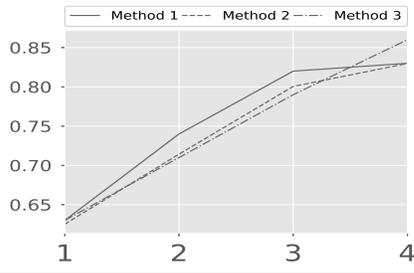
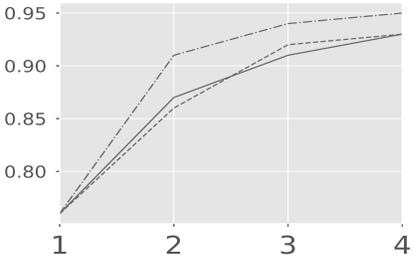
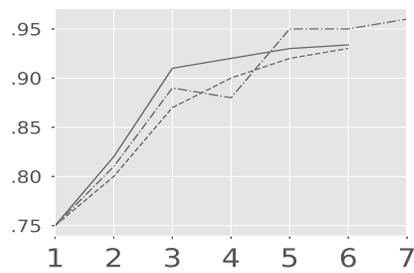
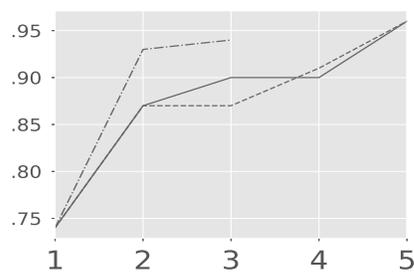
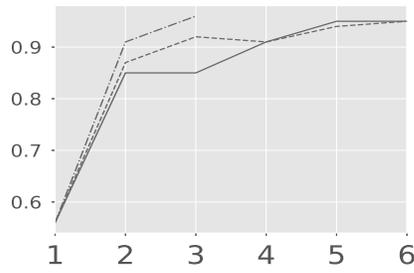
3 minutes timed-out) shows only 0.6% improvement for both objectives. While for the lower bounds for the smallest time step (2 minutes) show 6.6% improvement in objective 3.3.21, and 8% in objective 3.3.22. Nevertheless, our solution gives a value 20% lower over all step sizes for the second priority objective (objective 3.3.19). Thus, given the priority of the objectives, Method 3 dominates all the solutions studied produced using traditional discretisation, thus, producing near optimal solutions.

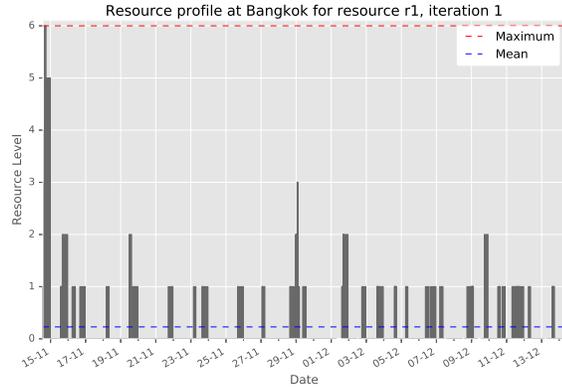
The dashed line in Figure 3.8, plotted against the right-hand y -axis, reveals that implementing a traditional time discretisation comes at a huge computational cost. Specifically, the discretisation with the smallest time step (2 minutes) is around 188 times slower than our solution. The reason why our method is significantly more efficient is because it only makes time intervals more granular when it is required and where the solution is more sensitive.

Table 3.5: Objective function breakdown for the Bangkok workshop. The best method in each iteration is highlighted in green.

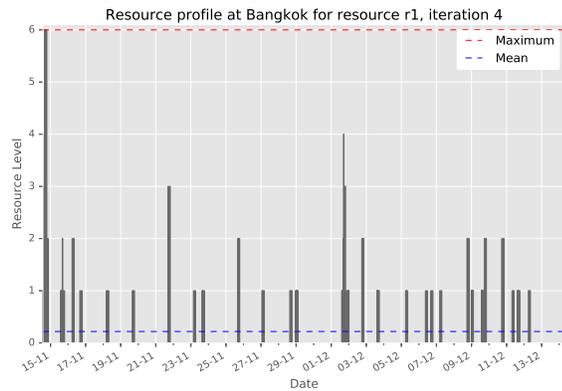
Iteration	Method / Obj.	Obj. 3.3.18	Obj. 3.3.19	Obj. 3.3.20	Obj. 3.3.21	Obj. 3.3.22	Obj. 3.3.23
1	Method 1	0	70	3	305	2184	54
	Method 2	0	70	3	305	2184	54
	Method 3	0	70	3	305	2184	54
2	Method 1	0	70	3	216	1559	62
	Method 2	0	70	3	209	1509	52
	Method 3	0	70	3	210	1523	60
3	Method 1	0	77	3	179	1304	66
	Method 2	0	77	3	179	1303	60
	Method 3	0	70	3	173	1258	60
4	Method 1	0	77	3	169	1231	71
	Method 2	0	77	3	169	1234	101
	Method 3	0	70	3	169	1233	55

Table 3.6: Comparison across 5 workshops during the interval splitting stage for the three different splitting methods.

Station	Accuracy Measure (per iteration)	Features	Splitting Methods		
			Method 1	Method 2	Method 3
Atlanta		Final # intervals	3343	3187	551
		Total CPU time (min)	60	52	4
Bangkok		Final # intervals	2713	2653	919
		Total CPU time (min)	105	95	72
Cairo		Final # intervals	3431	3117	349
		Total CPU time (min)	68	47	5.5
Dubai		Final # intervals	473	457	85
		Total CPU time (s)	6.6	4.4	0.36
Tokyo		Final # intervals	845	771	189
		Total CPU time (min)	234	72	3.3



(a) Resource profile for the first iteration.



(b) Resource profile for the last iteration.

Figure 3.7: Resource profiles for the Bangkok workshop for the first and last (4th) iterations of the interval splitting stage using Method 3.

3.5.2 Multi-workshop Case

To allow for interdependence between airlines and workshops, we consider several multi-workshop cases. Here, the algorithm solves the problem for all the workshops simultaneously. The first multi-workshop case considers four of the workshops used in the single workshop case. Using Atlanta, Cairo, Dubai and Tokyo, we produce a multi-workshop test set with 3002 flights and 167 aircraft, which produces 900 intervals. The interdependence for this case, due to their close proximity, is between

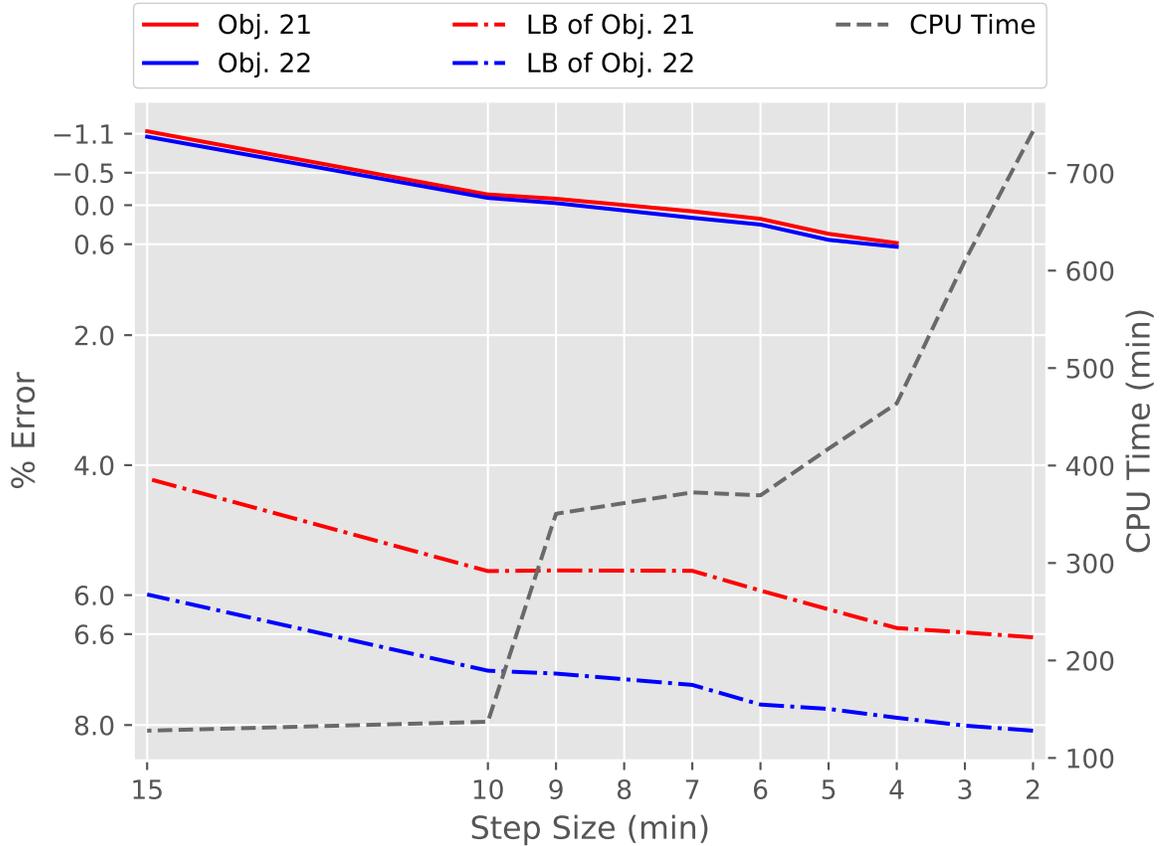


Figure 3.8: Result comparison (objectives 3.3.21, 3.3.22) for Method 3 vs traditional discretisation for different time steps.

Cairo and Dubai.

After a single iteration of the conflicting period selection, all violations are removed. Hence, the interval splitting stage begins. Table 3.7 shows, the intervals, computational time and accuracy measure value for each of the splitting methods. In each iteration, the best method is shown in green. As with the single workshop, this is determined by comparing objectives in decreasing order of priority until differing objectives are found, and one method presents an objective value lower than the rest. To avoid repetition, these have not been included due to their resemblance with the results in the single workshop case. The computational times reveal that Method 3

is the fastest (with a total of 1158 intervals and 9 minutes), followed by Method 2 (with a total of 4812 intervals and 58 minutes) and then Method 1 (with a total of 5020 intervals and 62 minutes). The highest accuracy measure, however, is no longer associated with Method 3, but with Method 1.

Table 3.7: Method comparison for the 4-workshop case.

Iteration	Method	Intervals	CPU time (s)	Accuracy Measure
1	Method 1	900	59	0.72
	Method 2	900	57	0.72
	Method 3	900	61	0.72
2	Method 1	1384	610	0.86
	Method 2	1390	414	0.83
	Method 3	1076	123	0.84
3	Method 1	2556	1252	0.93
	Method 2	2586	1258	0.92
	Method 3	1134	148	0.90
4	Method 1	5020	1777	0.99
	Method 2	4812	1739	0.98
	Method 3	1158	200	0.94

Given that it shows lowest solution times and objective values, provides a healthier fleet, and exhibits an acceptable accuracy measure we suggest that, for the multi-workshop case, the interval splitting stage should be implemented using Method 3. However, if more accuracy is sought, one can resort to Method 1 at the expense of higher computational times. When compared with the average accuracy measure obtained individually for the four workshops, Method 1 shows a 6% increase. This suggests that modelling interdependence between workshops, or, inter-airline cooperation, leads to more efficient allocation of resources.

Figure 3.10 shows the maintenance schedules produced using Method 3 for the four workshops involved in the multi-workshop test set. The x -axis represents the date and

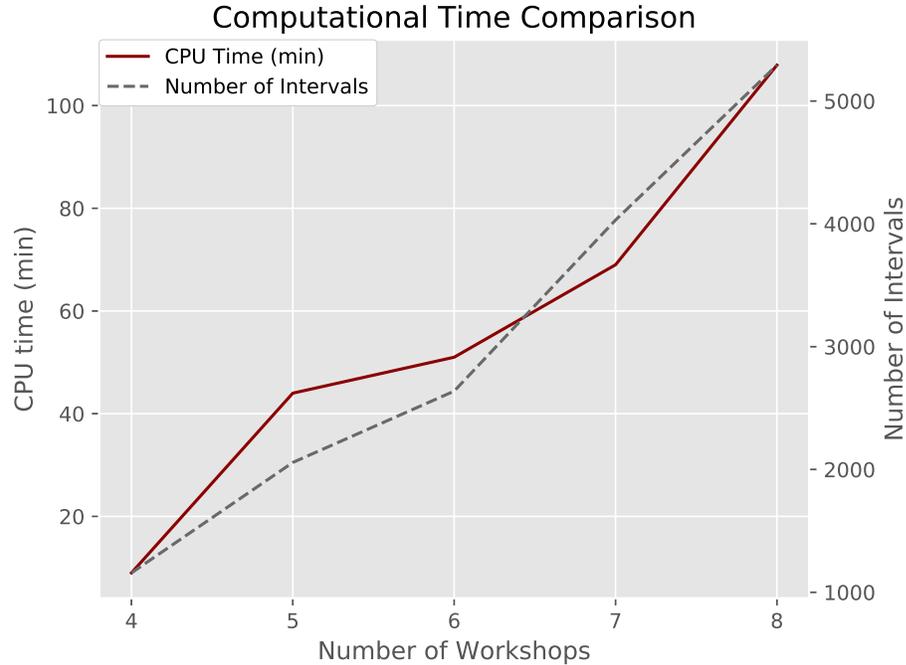


Figure 3.9: Computational comparison for Method 3 applied to different multi-workshop cases.

the y -axis shows each individual aircraft tail number. Hence, the plot shows when in the planning horizon a specific aircraft is being served, which type of maintenance it is receiving, and where the maintenance is taking place. Note that only one check 2 ($c = 2$ which corresponds to a C check) is scheduled in the busiest workshop, Atlanta, on the first day (14/12). The rest of the maintenance shown are checks $c = 1$ which correspond to A checks.

To study the scalability of the framework, four additional multi-workshop cases were created. They include, a 5 workshop case with 6871 flights and 216 aircraft, a 6 workshop case with 9681 flights and 260 aircraft, a 7 workshop case with 12402 flights and 429 aircraft, and, the largest, an 8 workshop case with 16000 flights and 529 aircraft. Additional to the five workshops considered in the single workshop case, the 8 workshop case considers workshops based in Abu Dhabi International Airport,

Beijing Capital International Airport, and, Madrid Barajas Airport. These cases were solved using Method 3 and until the algorithm was terminated; the least accuracy level achieved was of 0.97. The resulting computational comparison, in Figure 3.9, shows the computational times (red line on the left-hand y -axis) and final number of intervals (grey dashed line on the right-hand y -axis) for the different multi-workshop cases. The evolution of the CPU time and number of intervals seems to increase linearly with the number of workshops; the 8 multi-workshop case finalises with 5292 intervals and in 108 minutes. Therefore, it is possible that the framework is scalable for larger data sets. Clearly, this depends on the size and number of flights introduced each new workshop. Nonetheless, given the size of the largest instance considered, we have managed to demonstrate the computational efficiency and the potential of this framework.

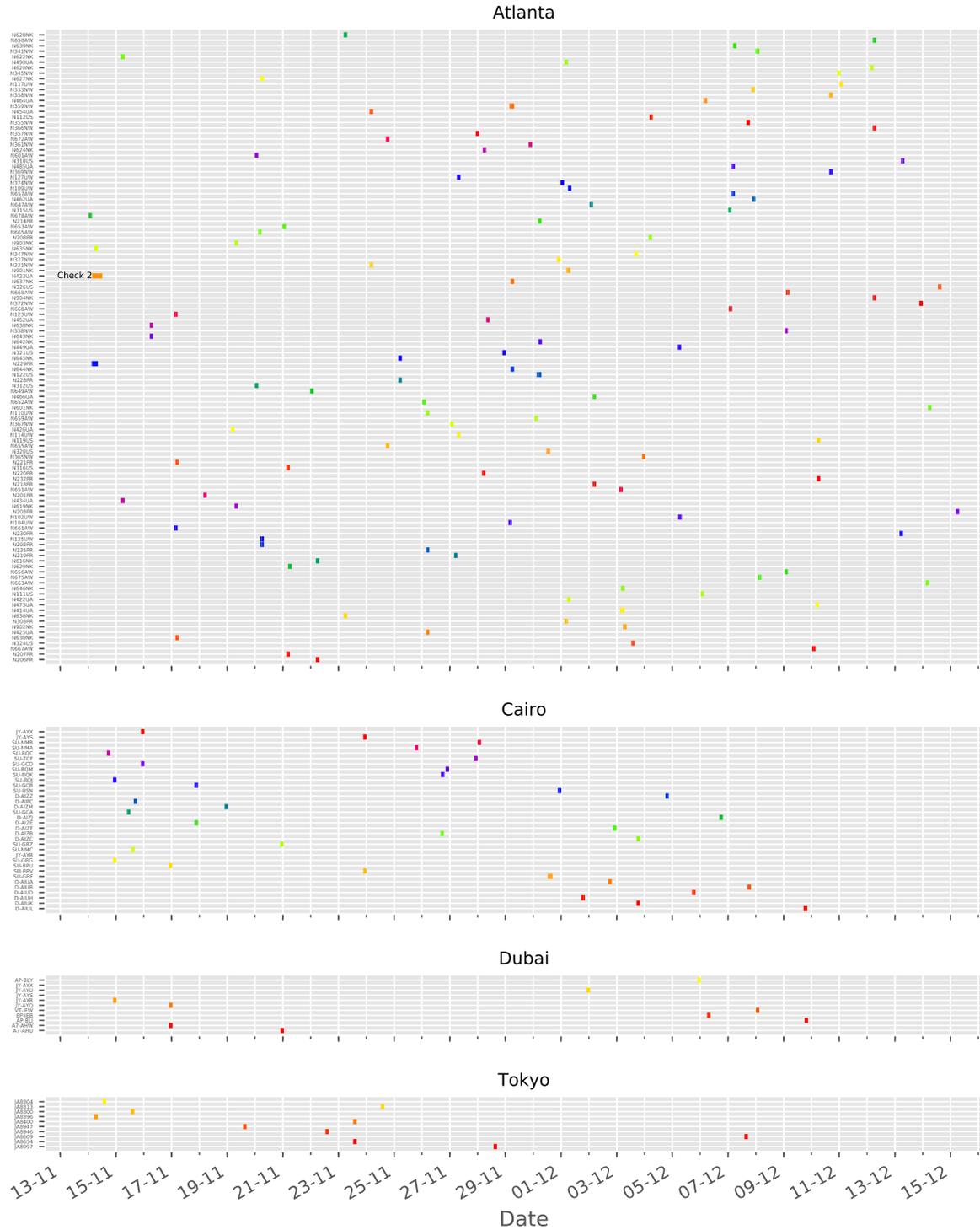


Figure 3.10: Maintenance schedules produced by using Method 3 for each aircraft and workshop in the first multi-workshop case.

3.6 Conclusions

We have solved the airline fleet maintenance scheduling problem considering tail assignment. Previous studies that tackle a similar problem have modelled a short-term planning horizon and tend to be computationally expensive even for moderately sized data sets. By using aircraft individual legal remaining flying hours and our interval based formulations, we tackle the problem while providing solutions in reasonable time. We present two multi-objective mixed integer linear programming formulations for this purpose. The first acts as a feasibility check for an input flight schedule and provides an initial set of variables for the tail assignment. If a feasible maintenance schedule was not found, the second formulation employs the location of the regulation violations to formulate a combined maintenance and tail assignment problem. This explores the different options across an aircraft journey and decides on the optimal allocation of flights, maintenance and resources. Additionally, our approach accounts for multiple resources and for generic types of maintenance.

In order to improve solutions, we implement a heuristic algorithm that consists of two stages: conflicting period selection and interval splitting. The conflicting period selection stage increases the size of the tail assignment problem gradually until we are able to produce a feasible maintenance schedule. After this, the interval splitting stage improves resource allocation.

Test results show that the algorithm is efficient, since it can solve large instances in reasonable computational time, for a 30-day planning horizon and provides good quality solutions. We solved a multi-workshop test case with 8 workshops, 16000

flights and 429 aircraft in under 2h. Highlighting the importance of developing new efficient formulations. Solutions present airlines with alternatives to their initial tail assignment during the planning stage. These solutions focus on satisfying maintenance regulations and keeping the aircraft healthy, while remaining commercially viable for the airlines. Results promote inter-airline cooperation, which allows for workshop resource sharing between airlines, since it provides a more efficient resource allocation per workshop.

Some limitations are worth noting. Stopping the iterative algorithm when the accuracy measure is maximal does not guarantee an optimal solution. Clearly, if continued indefinitely, the algorithm would reach the optimal solution, however, applying the stopping criterion gives a good quality solution in reasonable time. We compared the solution obtained using our method to the one obtained using a traditional discretisation with a small time step. This revealed that our solution is better and around 188 times faster, however, the theoretical proof remains to be done. Additionally, we have not considered the complications that may arise from some of the long-term maintenance types. In some cases, for example with life limited replacements, the inclusion of inventory control for spares and those being fixed would be paramount. Further work includes the implementation of a rolling horizon and the application of clustering for maintenance workshops with intersecting flights. Due to the lexicographic optimisation approach, the optimal solution(s) of the higher order objectives determine the optimal values of the lower order objectives. The solution space of the proposed formulation can be further expanded by relaxing the optimal value of the first priority objective, i.e. allowing for tolerance. An alternative way to

model the problem is to use a Goal Programming formulation and seek to minimise the weighted sum of the deviation from the optimum value of each objective. It is apparent that, the preferences of the decision maker(s) in terms of preference ordering (lexicographic optimisation) or weights (Goal Programming) will drive the outcome of the optimisation process. Future research may involve further experimentation of the decision maker(s) preferences in terms of the preferences of the decision maker(s).

Chapter 4

A Hybrid Solution Procedure for the Aircraft Recovery Problem with Operational Restrictions

4.1 Introduction

To encourage smooth running of airline operations, at the planning stage, airlines aim to create a schedule that suits them while respecting operational constraints. Even though, good pre-operational planning models exist, unplanned events often occur, affecting these plans. Hence, re-planning around the disruptions is paramount to resume ordinary operations (IATA, 2019). The airline recovery problem studies this problem and provides strategies to solve it. One the components of the airline recovery problem, at the operational stage, is the aircraft recovery (AR) problem. Other stages include, schedule, crew and passenger recovery (Clausen et al., 2010). The AR problem deals with the re-assignment of aircraft to flights, or flight legs, subject to operational restrictions, such that the cancelled and/or delayed flights are minimised. Operational restrictions may include, aircraft, maintenance, crew, and passenger disruption.

Table 4.1: Flight range according to flight length (Eurocontrol, 2011).

Flight Range	Aircraft Type	Distance	Time
Short-haul flight	Turboprop/narrow-body	≤ 1500 km	Under 3 hours
Medium-haul flight	Narrow/wide-body	≥ 1500 and ≤ 4000 km	3 to 6 hours
Long-haul flight	Wide-body	≥ 4000 km	6 to 12 hours

Aircraft types restrict the flights that certain aircraft are allowed to operate. There are three main types of aircraft: turboprop, narrow-body, and wide-body. The main difference among these is the passenger capacity and the flight range they operate. For instance, a turboprop aircraft, used for regional flights, is unsuitable to operate medium-haul flights. Convention for the classification of flights is given by the flight distance, or equivalently, the flight time. Flight range, and the link to aircraft types, is summarised in Table 4.1 which specifies the restriction for each aircraft type. Moreover, there are aircraft that operate mixed ranges, including: short/medium-haul, medium/long-haul, short/long-haul, and short/medium/long-haul; the last two are the rarest with a 0.2% and 0.9% of an international fleet (the sample included a mixed fleet from 67 airlines between 2008 and 2009, Eurocontrol, 2011).

Maintenance requirements are covered by several civil aviation authorities which provide a set of guidelines on how much an aircraft can be used between maintenance operations. Typically, regulations impose four medium/long-term airframe checks (A, B, C and D) and three long-term off-wing maintenance (engine, landing gear, and auxiliary power unit). These have to be performed after a certain number of flying hours, cycles, or months depending on the aircraft type (Cook and Tanner, 2008).

Crew restrictions are dominated by duty rules. Duty rules, as outlined by Barnhart et al. (2003), rely on regulations that balance working and resting hours for crew.

Firstly, crew duty periods (shifts) must end at the respective crew-base airport. The most important rules that crew duties must adhere to are, (1) the maximum duty length (i.e. time away from the crew-base), and (2) the maximum consecutive duty hours without breaks Barnhart et al. (2003); Maher (2015).

Passenger disruption restrictions are different for cancellations and delays. Under EU law, passengers have the right to some reparation (care, re-routing, or compensation) from airlines if certain thresholds are exceeded. As established by European Parliament and Council of the European Union (2004), for delays, passengers have the right to reparation if (i) their short-haul flight is delayed by two hours or more, (ii) their medium/long-haul flight within the EU, or their medium-haul flight outside the EU, is delayed by three hours or more, or (iii) their flight, not falling under (i) or (ii), i.e. other long-haul flights, is delayed by four hours or more. For cancellations due to “extraordinary” circumstances, airlines are exempt from providing passengers with reparation. Given that the disruptions considered for the AR problem are of this type, passenger restrictions for cancellations are not considered.

In this paper, we develop a generic framework that solves the AR problem by generating flight schedules that minimise flight cancellations and operational costs, subject to a number of operational constraints. This is done by using a new hybrid solution procedure with several components. We solve the AR problem using column generation. To generate columns, we model the subproblems using the shortest path problem with resource constraints (SPPRC). Such resource constraints allow us to model the different restrictions outlined above, namely, aircraft, maintenance,

crew and passenger delay, while keeping the algorithm computationally efficient. To solve each SPPRC we use a reinforced learning hyper-heuristic which selects, out of a set of metaheuristic algorithms, the most appropriate one to apply based on past performance. In addition, once the metaheuristics stop generating columns with negative reduced cost, we employ an exact bidirectional labelling algorithm with dynamic halfway point (Tilk et al., 2017) to guarantee optimal solutions for the LP relaxation of the AR problem. In order to generate upper bounds as well as lower bounds, we implement a diving heuristic to achieve close-to-optimal integer solutions.

The remainder of this paper is organised as follows: Section 4.2, discusses the relevant literature. Section 4.3 presents the proposed modelling approaches and the corresponding underlying concepts. Section 4.4 presents the solution methodology which includes preprocessing steps and a detailed breakdown of our hybrid solution procedure. Computational experiments are given in Section 4.5, while Section 4.6 summarises the conclusions and provides recommendations for future research directions.

4.2 Literature Review

The aircraft recovery (AR) problem deals with the operational assignment of aircraft to flights to recover from a so-called disruption. A disruption can be a series of unforeseen factors (e.g. air traffic controller strike, drone sighting) that leads to significantly reduced airline operations. Such assignment must, therefore, quickly propose a solution that minimises cancellations while respecting ordinary regulations and restrictions. Intrinsically linked to the AR problem, is the pre-operational planning stage

equivalent, the tail assignment (TA) problem (sometimes called aircraft rotation). It is a well-known combinatorial optimisation problem that deals with the assignment of individual aircraft to different flight legs without delay or cancellation considerations. Traditionally, there are three main types of formulation for these problems, **string-based** models, **time-space network** (TSN) models, and **multi-commodity network flow** (MCNF) models.

String-based models are a type of formulation that formulates the AR or TA problems using strings, i.e. sequences of connected flights that begin and end at a maintenance workshop, and that satisfy flow balance and maintenance regulations. Barnhart, Boland, Clarke, Johnson, Nemhauser and Shenoi (1998) introduced the use of string-based formulations for the TA problem. Some influential works employ the string-based formulation for the TA problem (Cohn and Barnhart, 2003; Sarac et al., 2006; Papadakos, 2009). Maher (2015) modelled the integrated airline recovery problem (including the AR problem) using a string-based formulation and solved it using a column-and-row generation algorithm.

Figure 4.1 shows a simple TSN, the blue arcs represent a flight path through the network: starting at airport A at time period 1, then flying to airport B at time period 2, then grounded at airport B until time period 3, and so on. The dashed arcs, represent non-scheduled connections. In TSNs, each airport is represented by a time line where nodes represent every departure/arrival at the corresponding airport time line. Arcs represent flights and connections between the different time lines. TSNs were first introduced by Jarrah et al. (1993) for the AR problem, and were made

known by Clarke et al. (1997) for the TA problem. TSN models have been widely applied (Thengvall et al., 2000, 2001; Løve et al., 2001; Hicks et al., 2005; Bratu and Barnhart, 2006; Orhan et al., 2012; Haouari et al., 2013; Liang and Chaovalitwongse, 2013; Marla et al., 2016; Safaei and Jardine, 2018; Khaled et al., 2018).

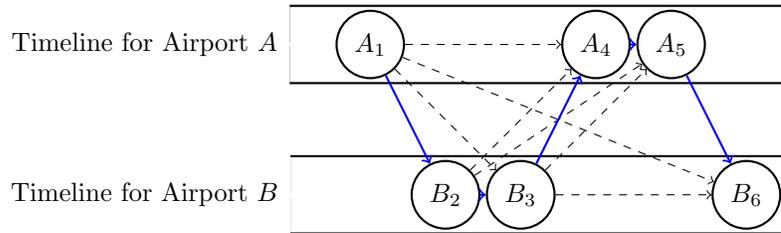


Figure 4.1: Time-Space Network (TSN) example (Torres Sanchez et al., n.d.).

MCNF models are based on a fleet-flow time-space network (layered TSN models), where each aircraft represents separate commodities and flow has to be preserved. The formulation, typically, includes capacity constraints (passengers and fleet) and conservation of aircraft, flight, and airport flow (Feo and Bard, 1989). The MCNF formulation, first employed for the AR problem by Argüello (1997), has been chosen by multiple authors (Bard et al., 2001; Yan and Tseng, 2002; Sriram and Haghani, 2003; Mercier et al., 2005; Eggenberg et al., 2010).

Varied solution approaches have been applied to solve the different formulations for the AR and TA problems. Column generation and branch-and-price, for instance, has been used by Barnhart, Boland, Clarke, Johnson, Nemhauser and Shenoi (1998); Grönkvist (2005); Sarac et al. (2006); Eggenberg et al. (2010); Maher et al. (2014, 2018); Liang et al. (2018), among others. Particularly, Liang et al. (2018) combined column generation with a basic multi-label algorithm (which solved each subproblem as a shortest path problem in a connection network) and applied it to solve the

AR problem with airport capacity constraints and maintenance considerations. Even though, this framework is computationally efficient, it is restrictive for further operational considerations. Several metaheuristic algorithms have also been used for the AR problem, greedy randomised adaptive search procedure (GRASP) (Argüello et al., 1997), local search (Løve et al., 2001), multi-objective genetic algorithm (Liu et al., 2008), and large neighbourhood search (Bisaiillon et al., 2011). The latter, algorithm that earned the authors first prize at the ROADEF challenge 2009 (Palpant et al., 2009), was applied to the aircraft and passenger recovery problem.

More generally, in the column generation literature, apart from the exact methods, such as branch-and-price, several heuristics have been developed to improve the quality of integer solutions without sacrificing computational time. Most of the column generation-based heuristics are of the following three types **restricted master**, **rounding**, and **diving** heuristics (Sadykov, Vanderbeck, Pessoa, Tahiri and Uchoa, 2019).

Restricted master heuristics rely on producing a restricted set of columns and solving the problem as a static IP. The effectiveness of this method relies on producing a set of feasible columns by applying heuristics to create feasible solutions. This method has been applied successfully in several problems, for instance vehicle routing problems (e.g. Agarwal et al., 1989; Bianchessi et al., 2014) and machine scheduling problems (Angelelli et al., 2014).

Rounding heuristics, consist on, after every iteration of the column generation procedure, rounding the values of the non-integer columns in the LP to their nearest

integer, while keeping the LP feasible. After this, heuristics are applied, typically to local search, to improve solutions. This approach has been implemented on, among others, some variants of the cutting stock problem (Cintra et al., 2008).

Diving methods are pure column generation based heuristics that consist on exploring the branch-and-bound tree by selecting a branch heuristically in a depth-first fashion. Hence, yielding fast primal solutions. In order to expand the search, some variants of the diving methods allow for a wider exploration of the branch-and-bound tree. This method has also been applied in variants of the cutting stock problem (Clautiaux et al., 2019), as well as the generalised assignment problem and the vertex colouring problem (both studied in Sadykov, Vanderbeck, Pessoa, Tahiri and Uchoa, 2019), and the vehicle routing problem (for instance, Pessoa et al., 2019). The authors of the latter paper, and their students, have open-sourced a related implementation of the algorithm (Marques et al., 2020), however, the functionality for solving the vehicle routing problems is currently unavailable as it is part of some other software from the same authors (Sadykov, Tahiri, Vanderbeck, Pessoa, Bulhões and Marques, 2019).

In the applied column generation framework, particularly in the scheduling related literature, the shortest path problem with resource constraints (SPPRC) is commonly employed to generate columns. It consists, as its name suggests, in finding, among all paths, the shortest path (from source to sink nodes) that satisfies a set of constraints for a defined set of resources. The SPPRC was first introduced by Desrochers (1986) as a subproblem for the bus driver scheduling problem and has, since then, been widely

studied and applied in a variety of settings including (but not limited to): the vehicle routing problem with time windows, the technician routing and scheduling problem, the capacitated arc-routing problem, on-demand transportation systems, and airport ground movement (Desrochers and Soumis, 1988; Feillet et al., 2004; Irnich and Villedieu, 2006; Righini and Salani, 2008; Bode and Irnich, 2014; Chen et al., 2016; Garaix et al., 2010; Tilk et al., 2017; Zamorano and Stolletz, 2017). Its ability to model complex cost functions as simple resource constraints and the variety of algorithms to solve it, make it a versatile and flexible method (Irnich and Desaulniers, 2005). For the AR problem, the idea of applying the SPPRC was previously sketched by Eggenberg et al. (2010). They solved the problem using column generation, where the subproblems were solved using the SPPRC. However, a single resource was included (to account for maintenance) and computational experiments appear to be incomplete (no computational times given).

Several algorithms have been developed to solve the SPPRC exactly. The most common exact algorithms are dynamic programming (DP) labelling algorithms. These methods build paths in a systematic fashion by starting at a source node and traversing the graph considering all feasible directions. Labels are created in order to efficiently compare different paths and discard the suboptimal ones. Irnich and Desaulniers (2005) presented an exact algorithm based on DP, the monodirectional forward labelling algorithm, based on the pioneering work by Desrochers and Soumis (1988). Boland et al. (2006) published a state augmenting algorithm that uses a monodirectional labelling algorithm to find an elementary path (one without repeating nodes). Righini and Salani (2006) introduced a bidirectional labelling algorithm for the SP-

PRC. The bidirectional algorithm is an extension of the monodirectional algorithm that supports search from both ends of the graph, hence, reducing the computational efforts. Moreover, they use bounding to mitigate *label explosion*. This typical phenomenon occurs due to the nature of the algorithm and the need to store the non-dominated labels of all partial paths. More recently, Tilk et al. (2017) released a bidirectional labelling algorithm with dynamic halfway point. Based on previous works (Righini and Salani, 2006; Pecin et al., 2017), the bidirectional search is bounded for both directions and these bounds are dynamically updated as the search in either direction advances. The algorithm has been shown to be significantly more efficient than monodirectional ones (Gschwind et al., 2018).

Even with some of the most recent algorithms, solving an instance of the SP-PRC exactly can be slow, thus, heuristic algorithms have been developed to find fast and promising shortest paths. These are mostly based on either DP or local search methods (metaheuristics). DP heuristics have been implemented by several authors. Feillet et al. (2004) presented a label correcting algorithm. Lozano et al. (2015) developed a pulse algorithm. Local search or metaheuristics start with a given path and perform a series of moves (node/arc: deletion, insertion, or exchange) to obtain another feasible path with lower cost. Metaheuristics implemented include Tabu search (Desaulniers et al., 2008), GRASP (Ferone et al., 2019), and hybrid particle swarm algorithm (Marinakis et al., 2017). Ferone et al. (2019) used GRASP in conjunction with a branch and bound scheme to solve the related: constrained shortest path *tour* problem.

When several low-level heuristics or metaheuristics are available to solve a certain problem, hyper-heuristic algorithms exploit the combination or selection of these to generate improved solutions. Figure 2.6 shows a scheme for a selective hyper-heuristic. Given some problem specific input and three different heuristics, in each iteration, the hyper-heuristic algorithm selects one heuristic (based on the ranking), applies it on the problem under consideration, evaluates its performance, and updates the ranks of all heuristics based on the performance. Given the scope of the area, we refer the reader to the recent and thorough review by Burke et al. (2013), and Burke et al. (2019). Specifically related to column generation, Li et al. (2015) combined a selection hyper-heuristic algorithm with column generation for the bus driver scheduling problem. To the extent of our knowledge, no other authors have combined hyper-heuristics with column generation.

The contributions of the present paper are (also outlined in Table 4.2),

- Hybrid Solution Procedure. We combine a reinforcement learning selection hyper-heuristic with column generation.
- Generic Modelling. The SPPRC allows for complex and generic operational constraints. We cater for aircraft, maintenance, crew duty, and passenger delay restrictions, but scope for more considerations is possible e.g. crew budget constraints, strategic delay costs, or airport capacity.
- SPPRC Algorithms. We implement two metaheuristic algorithms (two presented in this paper and one from the literature) and an exact algorithm (bidirectional labelling algorithm with dynamic halfway point; Tilk et al., 2017) to

efficiently solve instances of the SPPRC. These have been open-sourced as a Python package (Torres Sanchez, 2020).

- Computational Efficiency. Our hybrid solution procedure was tested on five different real-world tests instances with different sizes, and under different disruption scenarios; producing solutions in minutes.

Table 4.2: Summary of selected AR literature. Air. = aircraft type flight range considerations, Maint. = maintenance regulations considerations, Pass. = passenger delay restrictions considerations, SPPRC = use of shortest path problem with resource constraints, HH = use of hyper-heuristics, Opt. = close-to-optimal integer solution obtained, F = number of flights in largest test instance, A = number of aircraft in largest test instance, T = computational time of largest test instance.

Article	Air.	Maint.	Crew	Pass.	SPPRC	HH	Opt.	F	A	T
Argüello (1997)	×	×	×	×	×	×	×	42	16	10 s
Thengvall et al. (2000)	✓	×	×	✓	×	×	×	42	16	10 s
Thengvall et al. (2001)	✓	×	×	✓	×	×	×	1434	332	3.5 h
Løve et al. (2001)	✓	×	×	×	×	×	×	340	80	6 s
Bard et al. (2001)	×	×	×	×	×	×	×	162	27	3 min
Bratu and Barnhart (2006)	✓	×	✓	✓	×	×	×	1063	302	1.4 h
Liu et al. (2008)	×	×	×	×	×	×	×	140	19	n/a
Eggenberg et al. (2010)	×	×	×	✓	✓	×	×	608	85	50 min
Bisaillon et al. (2011)	×	✓	×	✓	×	×	×	1423	256	n/a
Maher (2015)	×	✓	✓	✓	×	×	✓	262	48	20 min
Liang et al. (2018)	×	✓	×	×	×	×	×	638	4	6 min
Present paper	✓	✓	✓	✓	✓	✓	✓	658	28	69 min

4.3 Modelling

To formulate the aircraft recovery (AR) problem, we introduce the following notation. Let \mathcal{J} be the set of all flights, indexed by j . Let \mathcal{K} be the set of aircraft, indexed by k . Let \mathcal{S} be the set of schedules, indexed by s , with each schedule having a certain cost, c^s , associated with it. Each schedule is composed of a series of flights, which can be denoted with \mathcal{J}^s , hence, $j \in \mathcal{J}^s$ means that flight j is present in schedule s . We can define the cost of a schedule as the sum of the costs of the flights that compose it. Flight costs take into account operational and delay costs; see Section 4.3.3 for more information.

Let the parameter d_{jk}^s , be equal to 1 if flight j is assigned to aircraft k in schedule s , and it is 0 otherwise. Finally, we define an assignment variable, x_k^s gets the value 1 if schedule s is assigned to aircraft k , while it is 0 otherwise.

4.3.1 Definitions

Sets

\mathcal{J} : Set of flights indexed by j , with $\mathcal{J} = \{1, \dots, J\}$;

\mathcal{K} : Set of all aircraft indexed by k , with $\mathcal{K} = \{1, \dots, K\}$;

\mathcal{S} : Set of schedules indexed by s .

\mathcal{J}^s : Set of flights in schedule s , with $\mathcal{J}^s \subseteq \mathcal{J}$;

Parameters

c^s : Cost of schedule s ;

d_{jk}^s : Parameter equal to 1 if flight j is assigned to aircraft k in schedule s , 0 otherwise.

Variables

x_k^s : 1, if aircraft k is assigned to schedule s , 0 otherwise.

4.3.2 Aircraft Recovery Problem Formulation

Using the notation defined, let us formulate the AR problem as follows,

Model 4.3.1. *AR using a generalised set covering formulation.*

$$\min \sum_s \sum_k c^s x_k^s \quad (4.3.1)$$

Subject to

$$\sum_s \sum_k d_{jk}^s x_k^s \geq 1 \quad \forall j; \quad (4.3.2)$$

$$\sum_s x_k^s \leq 1 \quad \forall k; \quad (4.3.3)$$

$$x_k^s \in \{0, 1\} \quad \forall k, s; \quad (4.3.4)$$

Objective 4.3.1 minimises the total cost of assigning aircraft to schedules. Constraints 4.3.2 ensure that each flight is assigned to at least one aircraft. Constraints 4.3.3 ensure that each aircraft is assigned to at most one schedule. Constraints 4.3.4 define the domain of the binary assignment variable.

To efficiently solve this problem, we can employ column generation. On the average case, column generation prevents from having to enumerate all schedules ($2^J - 1$).

4.3.3 Column Generation

To apply column generation, let us formulate a restricted and relaxed version of Model 4.3.1. For this purpose, we solve the problem on a reduced number of schedules. In every iteration, i , of the column generation scheme, we introduce a new schedule s , i.e. a column. The set of schedules \mathcal{S}^i is then updated as $\mathcal{S}^i = \mathcal{S}^{i-1} \cup \{s\}$ (for $i \geq 1$).

The corresponding schedule costs, c^s , are set appropriately (as discussed in Section 4.3.3). For the first iteration, we employ schedules \mathcal{S}^0 , where each schedule contains a single and distinct flight. That is, for $s \in \mathcal{S}^0$, $\bigsqcup_s \mathcal{J}^s = \mathcal{J}$. Additionally, to avoid these single flight schedules to appear in the solutions, we set each schedule cost, c^s , to a large flight cancellation cost. For a given iteration i of the column generation scheme, we can use schedules $s \in \mathcal{S}^i$, to formulate the restricted and relaxed version of Model 4.3.1 as follows,

Model 4.3.2. *Restricted and relaxed AR problem.*

$$\min \sum_s \sum_k c^s x_k^s \quad (4.3.5)$$

Subject to

$$\sum_s \sum_k d_{jk}^s x_k^s \geq 1 \quad \forall j; \quad (4.3.6)$$

$$\sum_s x_k^s \leq 1 \quad \forall k; \quad (4.3.7)$$

$$x_k^s \geq 0 \quad \forall k, s; \quad (4.3.8)$$

Aside from using a subset of schedules, the only other change with respect to Model 4.3.1, is the linear relaxation of the binary variable with constraints 4.3.8. Let π_j and μ_k be the dual variables for constraints 4.3.6 and 4.3.7, respectively. Hence, the reduced cost, for a given aircraft k , schedule s , is given as,

$$\bar{c}_k^s = c^s - \sum_j d_{jk}^s \pi_j + \mu_k . \quad (4.3.9)$$

Column generation involves solving a subproblem that will provide the column with the most negative reduced cost, or equivalently, finding a schedule s such that,

$$\bar{c}_k^* = \min_s \{\bar{c}_k^s\} = \min_s \left\{ c^s - \sum_j d_{jk}^s \pi_j \right\} + \mu_k . \quad (4.3.10)$$

Hence, generating a column for aircraft k corresponds to solving a subproblem and creating a schedule s with its corresponding flights. Specifically, each subproblem is modelled by employing the shortest path problem with resource constraints (SPPRC). Given that airlines' operational factors can be taken on an individual aircraft basis, we can model them in the subproblems as independent resource constraints.

Subproblems: Shortest Path Problem with Resource Constraints

To solve the subproblems for each aircraft k , given in Equation 4.3.10, we make use of the SPPRC. To model the subproblems accurately, let us define a network with some specific properties. Let $G = (N, A)$ be directed activity-on-arc or a TSN where the set of nodes N represents the airports under consideration at different time points, and the arcs, A , correspond to connections. Let G have a single source and sink nodes with no incoming or outgoing connections respectively. For two nodes $i, j \in N$ there is an arc, i.e connection, $g = (i, j) \in A$, if and only if it corresponds to a scheduled connection (flight or ground with $g \in \mathcal{J}$), or a non-scheduled connection ($g \notin \mathcal{J}$). We distinguish two types of non-scheduled connections: a **flight delay copy**, a delayed copy of a scheduled flight; and a **non-scheduled ground connection**. See Section 4.4.1 for more details.

For any connection g , let w_g denote the weight, orig_g and dest_g denote the head and tail nodes (respectively), a_g and d_g denote the arrival and departure times (respectively), r_g denote the flight range, and lastly, let T_g denote the type of connection (scheduled: flight or ground connection, non-scheduled: flight delay copy or ground connection).

Additionally, every connection has a corresponding operational cost. For a subproblem k and any scheduled connection, the operating cost is defined to be proportional to a standard operating cost c_k^{st} . Where c_k^{st} is either, the standard total operating cost (per unit time) for aircraft k , if the connection corresponds to a flight, or 0 otherwise. The costs associated with non-scheduled connections (per unit time) are, c_k^{od} for flight delay copies (operating costs under delay), and c_k^{gw} for ground connections (ground waiting costs). See Appendix B.2 for the values used in the computational tests.

To define resource consumption and constraints, let $\mathcal{R} = \{1, \dots, R\}$ be a set of resources. Let $\underline{\mathbf{L}} = (\underline{L}^1, \dots, \underline{L}^R)$ and $\bar{\mathbf{L}} = (\bar{L}^1, \dots, \bar{L}^R)$ be vectors for minimum and maximum resources, respectively. For each connection g , we denote the associated weight by w_g , and the resource consumption vector by $\mathbf{f}_g = (f_g^1, \dots, f_g^R)$. Each component in this vector is referred to as a resource extension function (REF) (Irnich and Desaulniers, 2005). For a given path, p , we denote the set of connections by $A(p)$, the weight of the path by $w(p)$, and the resource consumed along the path by $\mathbf{f}(p) = (f^1(p), \dots, f^R(p))$, where,

$$w(p) = \sum_{g \in A(p)} w_g \quad ; \quad f^r(p) = \sum_{g \in A(p)} f_g^r. \quad (4.3.11)$$

To define the weight w_g , for an arc g and subproblem k , we use,

$$w_g = \begin{cases} c_k^{\text{st}}(a_g - d_g) - \pi_g & \text{if } g \in \mathcal{J} \\ c_k^{\text{st}}(a_g - d_g) + c_k^{\text{od}}(d_g - d_{g'}) - \pi_j & \text{if } g \notin \mathcal{J} \wedge \text{COND}_0 \quad ; \\ c_k^{\text{gw}}(a_g - d_g) & \text{if } g \notin \mathcal{J} \wedge \text{COND}'_0 \end{cases} \quad (4.3.12)$$

where π_g is the dual variable for constraints 4.3.7. Here, condition $COND_0$ is true only if T_g corresponds to a flight delay copy of a scheduled flight g' , where,

$$g' = \sup \{g' \in \mathcal{J} : \text{orig}_{g'} = \text{orig}_g, \text{dest}_{g'} = \text{dest}_g, \text{ and } d_{g'} < d_g\}. \quad (4.3.13)$$

Condition $COND'_0$ is true only if T_g corresponds to a non-scheduled ground connection. Hence, Equation 4.3.12 defines the weights of scheduled and non-scheduled connections separately. For a given subproblem k and any scheduled connection g , the weight is proportional to the standard operating cost for aircraft k (c_k^{st}) times the duration of the connection, minus the corresponding dual value. Note that this may lead to negative weights. For a given subproblem k and any non-scheduled connection g , if the connection corresponds to a flight delay copy, then the weight is equal to the standard operating cost (c_k^{st}) times the duration of the connection, plus the operating cost under delay (c_k^{od}) times the delay with respect to the original flight, minus the dual value of the original flight. If the non-scheduled connection corresponds to a non-scheduled ground connection, then the weight is equal to the ground waiting cost (c_k^{gw}) times the duration of the connection.

Using these definitions, generating a column for each aircraft k corresponds to a schedule t produced by the connections of a path produced by solving an appropriate SPPRC. Such path is one that minimises delay while satisfying operational restrictions. That is, finding a path in G , p , which minimises $w(p)$ subject to resource constraints $\underline{\mathbf{L}} \leq \mathbf{f}(p) \leq \overline{\mathbf{L}}$, i.e. $\underline{L}^r \leq f^r(p) \leq \overline{L}^r$ for every resource r . To evaluate the column, the reduced cost can be calculated by updating the parameters d_{jk}^t , and using Equation 4.3.9. If the reduced cost is negative, the column is added with a cost

c^t . To minimise operational costs, c^t is set to be equal to the total operating costs of the schedule.

Resource Extension Functions

To take airline operational factors into account, we define appropriate resource extension functions (REFs) employed in the SPPRC. For this purpose, it is convenient to define some attributes for connections and aircraft. Each connection g , in a TSN, G , has attributes regarding the corresponding origin and destination airports (orig_g , dest_g), departure and arrival times (d_g , a_g), and flight range (r_g). Each aircraft k has an attribute r_k that represents the range that a certain aircraft type can operate. Let us define a set of seven resources,

$$\mathcal{R} = \{\text{mono}, \text{air}, \text{maint}, \text{crew-d1}, \text{crew-d2}, \text{pass}, \text{non-s}\},$$

corresponding to: an artificial monotone resource (required for the exact algorithm, see Section 4.4.3); operational restrictions: aircraft, maintenance, crew (duty restrictions 1 and 2 from Section 4.1), and passenger delay; and an additional restriction for non-scheduled connections.

We can define REFs for each resource. For extending partial path p_i (a path from source to node i) along arc $g = (i, j)$, resulting in partial path p_j (a path from source to node j passing through node i), we can set the resource consumption vector, $\mathbf{f}(p_j)$, according to the following REFs.

For the artificial monotone resource,

$$f^{\text{mono}}(p_j) = f^{\text{mono}}(p_i) + 1 \tag{4.3.14}$$

For aircraft restrictions,

$$f^{\text{air}}(p_j) = \begin{cases} 1 & \text{if not } COND_1 \\ 0 & \text{otherwise} \end{cases} ; \quad (4.3.15)$$

where condition $COND_1$ depends on the range of the aircraft under consideration, r_k . If r_k is short-haul or long-haul, then $COND_1$ is true only if r_g corresponds to the same range. If r_k is short/medium-haul, then $COND_1$ is true only if r_g is at most a medium-haul flight.

For maintenance restrictions,

$$f^{\text{maint}}(p_j) = \begin{cases} f^{\text{maint}}(p_i) & \text{if } \text{orig}_g = \text{dest}_g \\ 0 & \text{if } COND_2 \\ f^{\text{maint}}(p_i) + (a_g - d_g) & \text{otherwise} \end{cases} ; \quad (4.3.16)$$

where condition $COND_2$ is true only if the origin and destination airports, $\text{orig}_g = \text{dest}_g$, corresponds to a hub airport and the turnaround time, $a_g - d_g$, is at least long enough for the shortest type of maintenance allowed. We denote this quantity with `maintMin`.

For crew duty restrictions 1 and 2,

$$f^{\text{crew-d1}}(p_j) = \begin{cases} f^{\text{crew-d1}}(p_i) + (a_g - d_g) & \text{if } \text{orig}_g \neq \text{dest}_g \\ f^{\text{crew-d1}}(p_i) & \text{if not } COND'_2 \\ 0 & \text{if } COND'_2 \end{cases} ; \quad (4.3.17)$$

and

$$f^{\text{crew-d2}}(p_j) = \begin{cases} f^{\text{crew-d2}}(p_j) + (a_g - d_g) & \text{if } \text{orig}_g \neq \text{dest}_g \\ f^{\text{crew-d2}}(p_i) & \text{if not } COND_3 \\ 0 & \text{if } COND_3 \end{cases} \quad (4.3.18)$$

Where condition $COND_2$ is true only if the arrival and departure airport, $\text{orig}_g = \text{dest}_g$, corresponds to a hub airport. Condition $COND_3$ is true only if the origin and destination airports, $\text{orig}_g = \text{dest}_g$, corresponds to a hub airport and the turnaround time, $a_g - d_g$, is at least long enough for the shortest crew duty break. We denote this quantity with **breakMin**.

For the passenger delay restrictions,

$$f^{\text{pass}}(p_j) = \begin{cases} 1 & \text{if } COND_4 \\ 0 & \text{otherwise} \end{cases} ; \quad (4.3.19)$$

where condition $COND_4$ is true only if $g \notin \mathcal{J}$, corresponding to a flight delay copy of flight g' (defined by Equation 4.3.13), and either of the following hold: if r_g is short-haul, $d_g - d_{g'} \geq 2$; if r_g is medium-haul, $d_g - d_{g'} \geq 3$; or if r_g is long-haul, $d_g - d_{g'} \geq 4$.

Lastly, for non-scheduled connection restrictions,

$$f^{\text{non-s}}(p_j) = \begin{cases} 1 & \text{if } g \in \mathcal{J} \\ f^{\text{non-s}}(p_i) - 1 & \text{otherwise} \end{cases} . \quad (4.3.20)$$

With these definitions, we can set the minimum and maximum resource levels for

each resource as,

$$\underline{\mathbf{L}} = (0, 0, 0, 0, 0, 0, 0), \quad \overline{\mathbf{L}} = (|N|, 0, \text{FH}, \text{CD1}, \text{CD2}, 0, 1), \quad (4.3.21)$$

Recall that the entries are in the same order as the resources, \mathcal{R} . This enforces the following restrictions, for each resource r ,

$r = \text{mono}$: the artificial monotone resource increases by 1 every time an arc is traversed, with an upper bound equal to the number of nodes in the network;

$r = \text{air}$: aircraft can only perform flights that match with their type. Clearly, this can be included as part of the preprocessing stage, however, we incorporate it as this allows the modelling of more complex aircraft type constraints;

$r = \text{maint}$: maintenance increases with flying hours unless either the path extension corresponds to a ground connection or a maintenance stop is done, in which case it is set to 0. A maintenance stop is one done at a hub airport with enough time to perform at least the shortest maintenance. The upper bound, FH, corresponds to the maximum flying hours allowed by regulations (see Section 4.1);

$r = \text{crew-d1}$: crew duty rule 1, the maximum shift duration, must not exceed CD1 (see Section 4.1);

$r = \text{crew-d2}$: crew duty rule 2, the maximum flying hours without rest, must not exceed CD2 (see Section 4.1);

$r = \text{pass}$: passenger delay cannot exceed the limits imposed by regulations (see Sec-

tion 4.1);

$r = \text{non-s}$: enforces that no two consecutive non-scheduled connections are in a path.

A scheduled flight sets the resource to 1, otherwise, it is set to its previous value minus 1. Since the minimum for this resource is 0, and two consecutive non-scheduled connections will produce a value of -1, the required relation is enforced.

See Appendix B.2 for the values of FH, CD1, and CD2 used in the computational tests.

4.4 Solution Methodology

The hybrid solution procedure relies on several interacting components. Namely, the generation and update of a specific type of network and the combination of column generation with hyper-heuristics. In this section, we examine the different components of the hybrid solution procedure in detail. Particularly, we study the different algorithms employed in the column generation scheme, exact and metaheuristic, and adapt a reinforcement learning–great-deluge hyper-heuristic to create two column generation-specific hyper-heuristics.

4.4.1 Network Generation

Flight schedules allow us to construct the necessary TSN network for the subproblems. The TSN network, G , as defined in Section 4.3.3, is constructed as follows. A node is included for every airport under consideration at each different time point. We include two artificial nodes that represent the source and sink nodes, with no in-

coming or outgoing arcs respectively. Arcs are split by scheduled flights/ground arcs (those present in the initial flight schedule), and non-scheduled connections which we generate. Non-scheduled connections are generated between all airports with a prior existing flight (flight delay copies) and all ground arcs. Another type of non-scheduled connection are ferry or empty flights (sometimes positioning). However, ferrying is not always considered in the AR problem (Clausen et al., 2010), alternatively, some authors that account for crew recovery considered crew deadhead flights (where crew travel along with passengers), Maher (2015), for instance. Kenan et al. (2018) solved an extension to the TA problem (with maintenance) allowing ferry flights. They reported that the appearance of these types of connections in the solutions occurred only upon the decrease of the allocated ferrying costs (provided by an airline) and resulted in a small increase in profit. Thengvall et al. (2001) solved the AR problem using a TSN and also including ferry flights. Results showed that the number of ferry flights in the solution decrease as the recovery horizon increases. Additionally, the recovery horizons in this work were under 24 hours. Given that the recovery horizons considered in this research are at least 2 days, we do not account for ferrying; nonetheless, flight delays and cancellations are included.

Figure 4.2 shows a two airport example of a TSN with two different routes. Given the routes in the pre-operational flight schedules (blue and red arcs), non-scheduled connections are generated (dashed arcs) for flight delay copies and ground arcs. Moreover, due to airline restrictions, flight delay copies are not generated prior to the original flight. For instance, a copy for the blue flight B_3 to A_4 is only generated after, creating the flight delay copy B_4 to A_5 , but not B_2 to A_3 . This creates a cross-hatched

pattern between connected airports. Generating delay copies in this fashion, avoids creating additional arcs (as with most of the traditional methods), but fits more naturally into the original pre-operational plans. Clearly, larger networks benefit from more flight delay copies and therefore, more options when creating routes; hence, leading to better quality solutions. On the other hand, small networks provide less flight delay copies and typically correlate with worse quality solutions. See Section 4.5 for more information.

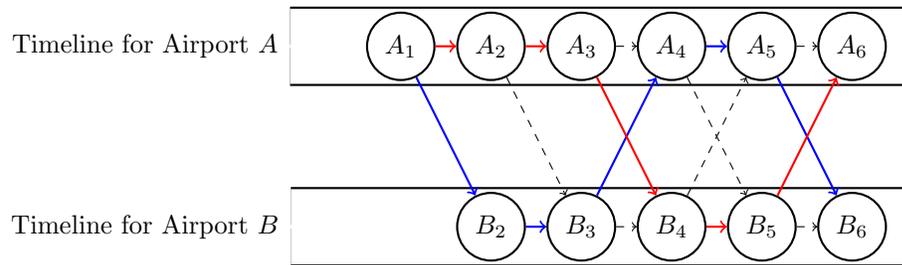


Figure 4.2: Generation of Time-Space Networks.

4.4.2 Hybrid Solution Procedure

There are several components in our hybrid solution procedure. Figure 4.3 outlines the three main parts. The TSN component (in yellow), as previously discussed in Section 4.4.1, generates the initial TSN based on an input flight schedule, then, once the dual values have been obtained from the solution of Model 4.3.2, we update the connection weights as discussed in Section 4.3.3. The column generation component (in purple), has three processes that involve, solving Model 4.3.2, generating a column and checking whether a proposed column has negative reduced cost. To solve the subproblems, we make use of the hyper-heuristic component of the solution procedure (in grey). Here, we select, out of a set of heuristics \mathcal{H} , an algorithm H to solve

the subproblem for aircraft k , hence, generating a column. Then, we evaluate the performance of the heuristic based on past performance and the cost of the column provided. If the column for aircraft k has negative reduced cost, then we add it to Model 4.3.2, apply the diving heuristic, and resolve the problem. Otherwise, we check whether all subproblems have been solved using an exact algorithm, A . If not, then we update the hyper-heuristic accordingly depending on which hyper-heuristic we are using. Broadly, this involves using the exact algorithm if all heuristics have produced a non-negative reduced cost for k , and updating the set of heuristics \mathcal{H} in different ways (see Section 4.4.3 for a more detailed explanation). If all subproblems have been solved using the exact algorithm and produced columns with non-negative reduced costs, we exit the loop.

We now proceed to describe the different components of the hybrid solution procedure. Particularly, we focus on the diving heuristic and the algorithms employed to solve the subproblems and on the hyper-heuristics used to select and evaluate the metaheuristic algorithms.

Diving Heuristic

We implement a diving method with limited backtracking from Sadykov, Vanderbeck, Pessoa, Tahiri and Uchoa (2019). This heuristic uses Limited Discrepancy Search (LDS) (introduced by Harvey and Ginsberg, 1995) to explore the branching tree at every iteration of the column generation procedure. The diving procedure involves recursively solving the LP relaxation of the master problem, and applying simple branching rules on the non-integer variables. The pseudo-code for the diving

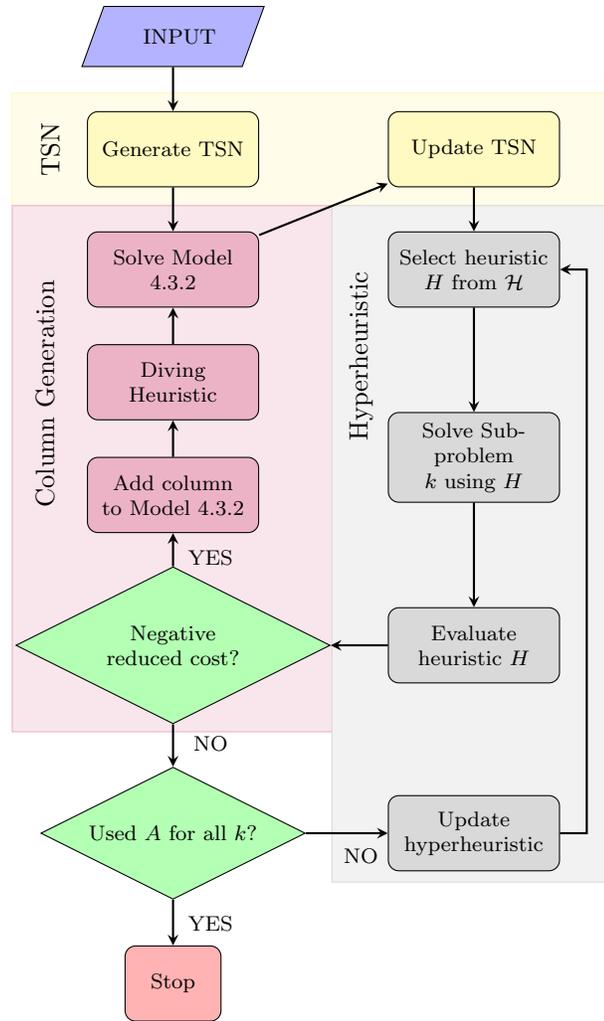


Figure 4.3: Hybrid Solution Procedure.

heuristic with LDS is shown in Algorithm 4. By using the LP relaxation, at any given iteration of the column generation procedure, and some algorithm specific parameter that determine the termination of the algorithm, the algorithm proceeds as follows. Two arrays are initialised, one to contain variables if they are fixed, `tabu`, and an integer variable, `depth`, to keep track of the number of iterations. After initialising an array for non-integer variables, line 4, the input LP relaxation is solved and all non-integer variables are extracted, lines 5 and 6. If some variables have already been fixed, we exclude them from the set of non-integer variables, line 7. If there are no

non-integer variables to be fixed then the solution to the LP relaxation is integer and the algorithm terminates, lines 8 and 9. Otherwise, the variable closest to 1 is fixed and added to the LP, line 11 and 12, the set of fixed variables is updated and so is the iteration count, lines 13 and 14. The process is repeated but now there is a chance that the resulting LP might be infeasible, this is caught by the if-statement in line 8, which results in algorithm termination. Additionally, if the number of variables fixed exceeds the input bound (`maxDiscrepancy`) or the number of iterations exceeds the input bound (`maxDepth`), the algorithm terminates.

Algorithm 4 Diving with LDS (Sadykov, Vanderbeck, Pessoa, Tahiri and Uchoa, 2019).

```

1: INPUTS: LP relaxation of the master problem, maxDiscrepancy, maxDepth.
2: Initialisations: tabu= [], depth= 0.
3: while |tabu| ≤ maxDiscrepancy and depth ≤ maxDepth do
4:   Set vars= []
5:   Solve the LP relaxation.
6:   Extract variables with non-integer values and add store them in vars.
7:   vars = vars \ tabu
8:   if vars is empty or the LP relaxation is infeasible then
9:     break
10:  end if
11:  Select the variable var ∈ vars with value is closest to 1.
12:  Update LP relaxation by fixing var to 1.
13:  Update tabu with var.
14:  depth = depth + 1
15: end while

```

4.4.3 Solving the Subproblems

In this section, we discuss different algorithms that can be employed to solve the subproblems. We provide four different algorithms to solve the SPPRC discussed in Section 4.3.3. Technically, all of these algorithms provide an elementary solution to the SPPRC, in other words, one without cycles. One exact algorithm, bidirectional

labelling algorithm with dynamic half-way point by Tilk et al. (2017) (Algorithm B.1.1), two metaheuristic algorithms introduced in this paper, Tabu search (Algorithm 6) and greedy elimination (Algorithm 7). Additionally, we adapted the metaheuristic GRASP from Ferone et al. (2019).

Exact Algorithm

To solve the SPPRC exactly, we implement a bidirectional labelling algorithm with dynamic halfway point (Tilk et al., 2017). It is an extension of the monodirectional algorithm that supports search from both ends of the graph, hence, reducing the computational efforts. The algorithm relies on two assumptions. Firstly, one of the resources must be a monotone resource. Such a resource is used for forward and backward searches, which we denote by *mono*. The assumption is easily met, as a monotone resource typically present in most applications e.g. time; if not, an artificial monotone resource can be created, e.g. the number of arcs visited. Second, that the REFs are invertible. These can be easily constructed for typical REFs (Irnich, 2008). Let, the inverse of the REF for a particular resource r , as introduced in Section 4.3.3, f_g^r , be denoted with b_g^r . We refer to these as the forward and backward REFs respectively. Moreover, we must have that the monotone resource is non-decreasing for forward REFs and non-increasing for backward REFs.

For this problem, the backward REFs can be defined using the REFs as per Section 4.3.3. Consider the extension of a *backward* partial path p_j (from the sink to node j) along arc $g = (i, j)$, resulting in backward partial path p_i (from the sink to node i passing through node j). The conditions used for the forward REFs are node-

invertable, hence, they do not vary with the direction that the arc is traversed. As they only depend on the attributes of each connection g which are invariable to the direction in which the arc is traversed. It is clear that, for any connection $g = (i, j)$ the attributes, origin-destination airport, flight arrival-departure times, and flight range, are mapped to the same value, regardless of the direction. Hence, all the forward REFs can be reused. Except, however, for the case of the monotone resource, `mono`, where, by requirement for the algorithm, we must have a non-increasing function. That is,

$$b^{\text{mono}}(p_i) = b^{\text{mono}}(p_i) - 1. \quad (4.4.1)$$

To apply the bidirectional algorithm, each node i has a series of forward and backward labels associated to it F_i^l and B_i^l , respectively. Each forward label takes the form,

$$F_i^l = (w(p_i^l), \mathbf{f}(p_i^l))$$

where p_i^l is the corresponding partial path from the source to node i . The first element, $w(p_i^l)$, is the weight of the partial path, and the second, $\mathbf{f}(p_i^l)$, is the resource consumption vector of the partial path.

Similarly, each backward label takes the form,

$$B_i^l = (w(p_i^l), \mathbf{b}(p_i^l))$$

where p_i^l is the corresponding backward partial path from the sink to node i . The elements of the label are the same as for the forward labelling, the weight of the backward partial path and the resource consumption vector (using the backward REF)

of the backward partial path.

Using these labels, dominance relations can be defined. Let F_i^l and F_i^m be two different forward labels for node i , with corresponding partial paths p_i^l and p_i^m . We say that label F_i^l dominates label F_i^m if $w(p_i^l) \leq w(p_i^m)$, $\mathbf{f}(p_i^l) \leq \mathbf{f}(p_i^m)$, and at least one the inequalities is strict. The vector inequality is satisfied if and only if $f^r(p_i^l) \leq f^r(p_i^m)$ for every resource r . Backward dominance relations are defined in a similar fashion, except for the monotone resource which is non-increasing, hence the inequality is flipped. That is, for two different backward labels B_j^l and B_j^m for node j (with corresponding partial paths p_j^l and p_j^m); we say that label B_j^l dominates label B_j^m if $w(p_j^l) \leq w(p_j^m)$, $b^r(p_j^l) \leq b^r(p_j^m)$ for all r except for $r = \text{mono}$, where $b^r(p_j^l) \geq b^r(p_j^m)$, and at least one the inequalities is strict.

The pseudo-code for the bidirectional labelling algorithm with dynamic halfway point (Tilk et al., 2017) is shown in Algorithm 5. The definitions of all the functions used can be found in Appendix B.1.1. The inputs required are, a graph (G), and lower and upper bounds for resources ($\underline{\mathbf{L}}$, $\overline{\mathbf{L}}$). The forward and backward halfway points (HF, HB) are set to $\underline{L}^{\text{mono}} = \text{HB} < \text{HF} = \overline{L}^{\text{mono}}$, line 2, as required for the search to have a dynamic halfway point. Further initialisations are, two sets of unprocessed paths (forward F, and backward B), and two sets of non-dominated, or “best”, paths (forward FB, and backward BB).

Once the direction is settled, in line 5 using `GETDIRECTION`, we proceed either a forward or backward search in a very similar manner as the monodirectional labelling algorithm. After choosing a path from the unprocessed set of paths, lines 7 (forward)

or 17 (backward), we check if the halfway point has been reached, lines 8 or 18, if not, then we proceed to extend the selected path in the traditional way, lines 9-13 or 19-23. After this, we update the value of the halfway point to reduce the size of the search in the opposite direction. That is, in the forward search, in line 14, we update HB to a higher value if we have searched past it. Conversely, in the backward search, in line 24, we update HF to a lower value if we have searched past it. Moreover, in lines 15 and 26, at the end of every iteration, we can apply some dominance rules to remove unnecessary paths. The function CHECKDOMINANCE, in turn, updates the lists of non-dominated or “best” paths FB and BB. Alternatively, as the original authors of the algorithm suggest, this function can be called more sparingly. Additionally, in this step, we make use of a powerful property, that boosts the performance of the algorithm. For any pair of comparable paths (ending on the same node), Q and Q' , if Q is proved to dominate Q' , then Q' and all of its extensions, do not have to be explored. The reason for this is that no Pareto-optimal extension of Q' exist that cannot be produced from extensions of Q (Irnich and Desaulniers, 2005). Please note that this property does not hold for the elementary version of the SPPRC.

The algorithm carries on like this, updating the direction of search in every iteration, and terminating when no more paths remain to be processed in any direction. Once all the labels have been processed, labels have to be joined to produce valid paths. Originally, this was done using the procedure “Join” (Algorithm 3) from Righini and Salani (2006), which consists on merging all compatible forward and backward labels to find the optimal path. The algorithm relies on an additional “halfway check”, from the same paper, which ensures that the merged labels are over the halfway point.

In line 32, with JOIN, we use a simplified version that uses the lists of non-dominated forward and backward paths.

Algorithm 5 Bidirectional labelling algorithm with dynamic halfway point for the SPPRC (Tilk et al., 2017).

```

1: INPUTS:  $G = (V, A)$ ,  $\underline{L}$ ,  $\overline{L}$ ;
2: Initialisations:  $HF = \overline{L}^{\text{mono}}$ ,  $HB = \underline{L}^{\text{mono}}$ ,  $F = \{(s)\}$ ,  $B = \{(t)\}$ ,  $FB = \emptyset$  and  $BB = \emptyset$ 
3:  $\triangleright$  See Appendix B.1.1 for functions employed
4: while  $F \neq \emptyset$  or  $B \neq \emptyset$  do
5:   direction = GETDIRECTION( $F$ ,  $B$ )
6:   if direction = forward then  $\triangleright$  Forward labelling
7:     Choose a path  $Q \in F$ , with  $h(Q) = i$ ; remove  $Q$  from  $F$ ;
8:     if  $Q^{\text{res}} \leq HF$  then
9:       for  $(i, j) \in A$  do
10:        Extend path  $Q$  along  $(i, j)$ , denoted by  $(Q, j)$ ;  $\triangleright$  Path extension step
11:        if  $(Q, j)$  is feasible then Add  $(Q, j)$  to  $F$ ;
12:        end if
13:      end for
14:       $HB = \max\{HB, \min\{Q^{\text{res}}, HF\}\}$ 
15:      CHECKDOMINANCE( $Q$ , forward,  $F$ );
16:    end if
17:  else if direction = backward then  $\triangleright$  Backward labelling
18:    Choose a path  $Q' \in B$ , with  $t(Q') = j$ ; remove  $Q'$  from  $B$ ;
19:    if  $Q'^{\text{res}} > HB$  then
20:      for  $(i, j) \in A$  do
21:        Extend path  $Q'$  along  $(i, j)$ , denoted by  $(Q', i)$ ;
22:        if  $(Q', i)$  is feasible then Add  $(Q', i)$  to  $B$ ;
23:        end if
24:      end for
25:       $HF = \min\{HF, \max\{Q'^{\text{res}}, HB\}\}$ 
26:      CHECKDOMINANCE( $Q'$ , backward,  $B$ );
27:    end if
28:  else  $\triangleright$  If no direction
29:    break
30:  end if
31: end while
32: JOIN( $FB$ ,  $BB$ )

```

Metaheuristic Algorithms

We present three metaheuristic algorithms to use in the hyper-heuristic framework. These include: two tailor made algorithms developed by us, a Tabu search algorithm (inspired by Desaulniers et al., 2008) and a greedy elimination algorithm; and one recent algorithm from the literature, a greedy randomised adaptive search procedure (GRASP), adapted from Ferone et al. (2019). Our algorithms exploit the speed provided by a standard shortest path algorithm. The underlying reasoning is that obtaining the shortest path using a standard algorithm on a modified network that forces a resource feasible path, is computationally cheaper than using an SPPRC specific algorithm. All algorithms have the following common inputs: a TSN, G (as defined in Section 4.3.3), and lower and upper bounds for resources \underline{L} and \bar{L} (as defined in Section 4.3.3).

The first metaheuristic algorithm we have developed, is a Tabu search algorithm. The pseudo-code is provided in Algorithm 6, and the definitions of all the functions used can be found in Appendix B.1.2. In addition to the common inputs, it requires a user defined large number, M . After some appropriate initialisations, we proceed with the algorithm until a resource feasible path is found. First, in line 6, we find a partial shortest path, p , between the node `neighbour` and the sink, using a standard shortest path algorithm. Given the initialisations, the first path obtained is one from source to sink. If the partial path is valid, then we update the full path (from source to sink), `path`. This is done by merging the full path with the partial path after the branching point (the node `neighbour`). If the resulting full path is resource feasible,

we stop the algorithm; otherwise, we find the edge that makes the full path resource infeasible, line 12, and place a tabu there, line 13. This is done by setting the weight of the edge to be M , hence, is avoided by the shortest path algorithm. Alternatively, one can remove the edge and add it back again once a new tabu edges is produced.. Using the new tabu edge, we update the `neighbour` node as the head of the tabu edge, line 18. The function in line 13, `GETNEIGHBOUR`, ensures that the tabu is not placed on the same edge twice. Instead, it backtracks from the current tabu edge, returning the backward neighbouring edge (one incident to the tabu edge) with the largest weight. Please see Appendix B.1.2 for more details on the functions used. If the operations performed results in a non-valid partial path, then, in line 16, we place the tabu edge on a backward neighbouring edge of the current tabu edge.

Algorithm 6 Tabu Search for the SPPRC.

```

1: INPUTS:  $G = (V, A)$ ,  $\underline{L}$ ,  $\overline{L}$ .
2: Initialisations:
3:  $\text{stop} = \text{False}$ ,  $\text{edgesToCheck} = A$ ,  $\text{neighbour} = \text{Source}$ ,  $\text{tabu} = \text{None}$ ,  $\text{path} = []$ ,  $\text{neighbourhood} = []$ .
4:  $\triangleright$  See Appendix B.1.2 for functions employed
5: while  $\text{stop}$  is False do
6:    $p = \text{SHORTESTPATH}(\text{neighbour}, \text{Sink}, G)$ 
7:   if  $p$  is valid then
8:      $\text{path} = \text{UPDATEPATH}(\text{neighbour}, \text{neighbourhood}, p)$ 
9:     if  $\text{path}$  is resource feasible then
10:       $\text{stop} = \text{True}$   $\triangleright$  Terminate the algorithm
11:     else
12:       Find edge  $g$  that makes the path resource infeasible
13:        $\text{tabu}, \text{edgesToCheck}, \text{neighbourhood} = \text{GETNEIGHBOUR}(\text{edgesToCheck}, \text{neighbourhood}, g)$   $\triangleright$  Update tabu edge, array of edges to check and neighbourhood
14:     end if
15:   else
16:      $\text{tabu}, \text{edgesToCheck}, \text{neighbourhood} = \text{GETNEIGHBOUR}(\text{edgesToCheck}, \text{tabu})$   $\triangleright$  Update tabu edge, array of edges to check and neighbourhood
17:   end if
18:    $\text{neighbour} = h(\text{tabu})$   $\triangleright$  Update neighbour as head node of current tabu edge
19: end while

```

The second metaheuristic algorithm we have developed, the Greedy Elimination algorithm, is shown in Algorithm 7. Using the common inputs, and after the appropriate initialisations, in line 5, a standard algorithm is applied to obtain a full path, p , from source to sink. Please see Appendix B.1.2 for more details on this function. If the full path is valid, then we check if it is resource feasible. If this is true, then we terminate the algorithm, line 8; otherwise, we remove the problematic edge (the edge that makes the path resource infeasible) and resolve the shortest path problem. If the resulting path turns out to be non-valid, we add back the edge we previously removed, and instead, remove a backward neighbouring edge of the edge we previously

removed. We continue in this fashion until a resource feasible path is obtained.

Algorithm 7 Greedy Elimination Algorithm for the SPPRC.

```

1: INPUTS:  $G = (V, A)$ ,  $\underline{L}$ ,  $\overline{L}$ .
2: Initialisations:  $\text{stop} = \text{False}$ ,  $\text{neighbourhood} = []$ .
3:  $\triangleright$  See Appendix B.1.2 for functions employed
4: while  $\text{stop}$  is  $\text{False}$  do
5:    $p = \text{SHORTESTPATH}(\text{Source}, \text{Sink}, G)$ ;
6:   if  $p$  is valid then  $\triangleright$  If a path exists
7:     if path is resource feasible then
8:        $\text{stop} = \text{True}$   $\triangleright$  Terminate the algorithm
9:     else
10:      Find edge  $g$  that makes the path resource infeasible
11:      Remove  $g$  from  $G$ 
12:    end if
13:  else  $\triangleright$  If no path exists
14:    Add  $g$  back to  $G$ 
15:     $\triangleright$  Get new neighbouring edge to remove and update neighbourhood array
16:     $\text{edge}, \text{neighbourhood} = \text{GETNEXTNEIGH-}$ 
     $\text{BOURINGEDGE}(\text{neighbourhood}, g)$ 
17:    Remove  $\text{edge}$  from  $G$ 
18:  end if
19: end while

```

We implemented an adapted version of the GRASP presented by Ferone et al. (2019). The inputs required, apart from the common inputs, include: `maxiter`, the maximum number of iterations to terminate the algorithm; `maxiterLocal`, the maximum number of iterations to terminate the local search algorithm; α , the greediness factor (a value close to 1 represents a greedy selection, and a value close to 0 represents to a random selection); c , a cost function to evaluate solutions given by the algorithm; and M a large number to use as a penalty. Solutions are defined as ordered sets of nodes in the network, connected or not. The cost associated with each solution reflects on the connectivity and total weight of the path provided by the sequence of nodes. That is, for a potential path (any set of two or more nodes in the graph) in a

given solution, the associated cost is equal to the sum of the weights of the edges in the potential path. However, if the potential path is not valid, then the cost function returns the penalty value, M .

After the appropriate initialisation, until the number of iterations exceeds `maxiter`, the algorithm generates a solution, line 5, improves it using local search, line 6, evaluates and updates the best solution according to whether there is an improvement, lines 7 and 8. The functions in lines 5 and 6, defined in Appendix B.1.3, perform various operations.

The `CONSTRUCTSOLUTION` function, line 5, after sampling a random starting node, constructs a new solution by iteratively appending nodes based on their cost. To select a node to append to the solution, we compute a *restricted candidates list*, with all the nodes such that their cost, with respect to the last node in the solution, satisfies a simple equation involving the greediness factor α . By the definition of the cost function, if no edge exists between a candidate node and the last node in the solution, then the corresponding cost will be the penalty M . The node to append to the solution is randomly sampled from the restricted candidates list. This process is iterated until the solution contains all the nodes in the graph.

The `LOCALSEARCH` function, line 6, improves the current solution constructed in line 5, `solution`, by identifying paths using the nodes in it. Precisely, we try to identify a valid path using a random sized sample of the permutations of the nodes in the solution. The new path is a candidate solution. As previously, if the candidate solution is a non-valid path, then this is reflected in the cost. If the candidate solution has a lower cost than the current solution and is a resource feasible path, we replace

the current solution with the candidate solution. This process is repeated until the number of local iterations exceeds `maxiterLocal`, returning the updated (or not) solution.

Algorithm 8 Adapted GRASP for the SPPRC (Ferone et al., 2019).

```

1: INPUTS:  $G = (V, A)$ ,  $\underline{L}$ ,  $\overline{L}$ , maxiter (# of iterations), maxiterLocal (# of
   local search iterations),  $\alpha$  (Greediness factor),  $c$  (cost function),  $M$  (penalty).
2: Initialisations: stop = False, iter = 0, solution = [ ], best = [ ]
3:  $\triangleright$  See Appendix B.1.3 for functions employed
4: while iter < maxiter do
5:   solution = CONSTRUCTSOLUTION( )
6:   solution = LOCALSEARCH(solution)
7:   if  $c(\text{solution}) < c(\text{best})$  then
8:     best = solution
9:   end if
10:  iter = iter + 1  $\triangleright$  Increment iteration counter
11: end while

```

Hyper-heuristic Algorithm

We employ the reinforcement learning – great-geluge hyper-heuristic proposed by Özcan et al. (2010). Algorithm 9 outlines the approach. It relies on the use of utilities to select heuristics from a set of heuristics \mathcal{H} . In every iteration, we select the heuristic with highest utility, apply it, and evaluate it using a function $f(\cdot)$, which we are trying to minimise. Hence, the solution is compared to the previous one, f_p , to identify an improvement. If the heuristic has led to an improvement, i.e. a lower value with respect to the previous solution, we improve the utility for that heuristic, if not, then we worsen it. The authors define three different functions that can be employed to update the utilities: addition, division or square root. That is, for a heuristic $H_i \in \mathcal{H}$ the corresponding utility, u_i , is worsened by either $u_i = u_i - 1$ (U1), $u_i = u_i/2$ (U2), or $u_i = \sqrt{u_i}$ (U3), and improved only for the additive case, by $u_i = u_i + 1$. The

improvement rate of U1 causes it to provide less exploration of the heuristics than the other utilities, favouring better performing heuristics Özcan et al. (2010).

Algorithm 9 Reinforcement Learning – Great-Deluge Hyper-heuristic (Özcan et al., 2010)

```

1: INPUTS:  $\mathcal{H}$  (set of heuristics), utilities (array with initial utilities),
    $f(\cdot)$  (function to evaluate heuristic).
2: Initialisations: Apply a random heuristic  $H$  and set  $f_p = f(H)$ 
3: while Stopping criteria not satisfied do
4:   Choose heuristic  $H \in \mathcal{H}$  with maximum utility
5:   Apply heuristic  $H$ 
6:   if  $f(H) < f_p$  then
7:     Update utilities by improving the utility of heuristic  $H$ 
8:   else
9:     Update utilities by worsening the utility of heuristic  $H$ 
10:  end if
11:   $f_p = f(H)$ 
12: end while

```

To adapt the great-deluge hyper-heuristic for our hybrid solution procedure, we consider some column generation specific criteria. One hyper-heuristic is created per subproblem which selects a heuristic and uses it to solve the subproblem, hence, generating a column. The evaluation function f is defined to be the reduced cost produced by the column. In order to update the utilities, we compare the reduced cost with the previous reduced cost for the same subproblem. The utilities are then updated as outlined above, being shared across all subproblems. Moreover, we can update the set of heuristics and apply the exact algorithm in different ways. We outline two hyper-heuristic algorithms that use this column generation specific criteria, HH1 and HH2.

In HH1, for a particular subproblem, we apply all heuristics, selecting them based on their utility, until either, a column with negative reduced cost is generated, or all

heuristics fail to generate a column with negative reduced cost. If all heuristics fail, we apply the exact algorithm to generate a column. If the column has a negative reduced cost, we repeat the process until the exact algorithm generates columns with non-negative reduced cost for all subproblems.

In HH2, for a particular subproblem, k , we define a subset of heuristics $\mathcal{H}_k \subseteq \mathcal{H}$. A heuristic is selected from \mathcal{H}_k , based on the utilities, and applied to the subproblem. If the selected heuristic produces a column with non-negative reduced cost for a subproblem k , we remove it from \mathcal{H}_k . Thus, it cannot be selected for future subproblems k . Once all heuristics have been removed from \mathcal{H}_k (i.e. all heuristics have generated columns with non-negative reduced cost) we use employ the exact algorithm to generate columns for k thereafter.

ure

4.5 Computational Experiments

The hybrid solution procedure was coded in Python, using the PuLP interface and the open-source solver Cbc (johnjforrest, Vigerske, Gambini Santos, Ralphs, Hafer, Kristjansson, jpfasano, EdwinStraver, Lubin, rlougee, jpgoncal1, h-i-gassmann and Saltzman, 2020). As already mentioned, we have released the implementation of the SPPRC algorithms used (metaheuristic and exact) as a freely available Python package, `cspy` (Torres Sanchez, 2020). The package includes the implementation of the hybrid particle swarm algorithm for the SPPRC by Marinakis et al. (2017); however, this algorithm could not cope with our test instances.

To test the hybrid solution procedure which involves a hyper-heuristic algorithm

for column generation, as outlined in Section 4.4.2, we generated five different test instances. For this purpose, we obtained flight schedules from Flightradar24 AB (2018) using the package `pyflighdata` (Allamraju, 2014). After an extended data gathering stage, we can easily filter data per airline. The five test instances were constructed with flights in operation during varying recovery horizons in February 2017.

Table 4.3 specifies the following information about the instances: the number of flights, the number of aircraft, the number of aircraft types, the recovery horizon, the dates of the instance, and the size of the TSN generated (as outlined in 4.4.1). Table 4.4 summarises the algorithms used and their parameters, these are as suggested by their respective authors (H3 from Ferone et al., 2019; DH from Sadykov, Vanderbeck, Pessoa, Tahiri and Uchoa, 2019). Please refer to Appendix B.2 for more information on the composition of the fleet and specific parameters employed in the computational tests.

Table 4.3: Test instances.¹ F = # of flights, A = # of aircraft, T = # of aircraft types, H = length of recovery horizon in days, TSN= size of time-space network (# of nodes, # of arcs).

Name	F	A	T	H	Dates	TSN
I1	212	6	2	10	16/02/2017 - 26/02/2017	(220, 800)
I2	273	6	1	10	17/02/2017 - 27/02/2017	(277, 992)
I3	230	8	1	2	17/02/2017 - 19/02/2017	(240, 909)
I4	151	2	2	8	17/02/2017 - 25/02/2017	(168, 582)
I5	187	8	3	7	14/02/2017 - 21/02/2017	(220, 794)

¹ The data was obtained from real-world airlines with their respective hub airport(s). For data protection, we have removed the airline names.

The computational tests are split into two sections. Firstly, we explore the performance of the two different hyper-heuristics proposed and the different utility functions.

Table 4.4: Algorithms used in solution procedure.

Abbrev.	Algorithm	Parameters
A1	Bidirectional labelling (Algorithm 5)	n/a
H1	Tabu Search (Algorithm 6)	n/a
H2	Greedy Elimination (Algorithm 7)	n/a
H3	GRASP (Algorithm 8)	<code>maxiter</code> = 100, <code>maxiterLocal</code> = 10, α = 0.2, M = 10^{10}
DH	Diving Heuristic (Algorithm 4)	<code>maxDepth</code> = 3, <code>maxDiscrepancy</code> = 2

To do so, we employ a base disruption case where a single aircraft is grounded for 24 hours. After this, and by employing the faster hyper-heuristic set up, we study the effects of varied types of disruption across the different instances. Particularly, two types of disruptions are studied, up to five aircraft are delayed or cancelled, and airport closure (temporary or for the duration of the recovery period). The duration of these disruptions ranges from 24 to 144 hours (or the end of the planning horizon).

4.5.1 Analysis of Hyper-heuristics

In Section 4.4.3, we introduced two variations of the hyper-heuristic component for our hybrid solution procedure, we named them HH1 and HH2. In short, the main difference is that HH2 stops using a certain metaheuristic as soon as they produce a column with a non-negative reduced cost; while HH1, does not have this restriction. In this section we explore the performance of these two variants using the base disruption case where a single aircraft is grounded for 24 hours. Table 4.5 shows the counts for the different metaheuristic algorithms as chosen by each of the hyper-heuristics (HH1 and HH2) with the three different utility functions U1 (additive), U2 (division), and U3 (square root). The average time per column, i.e. per subproblem, is also included.

The counts shown by HH1 are higher given that it provides a fairer exploration of the metaheuristics, even more so using utilities U2 and U3 (Özcan et al., 2010). Algorithm H1 (Tabu Search) dominates in both of the hyper-heuristics and the three utility functions. Even for the largest instance, I2, the average computational time per iteration is under 1 second. Algorithm H2 (Greedy Elimination) also shows small computational times, regardless of the size of the instance. Even though it takes significantly longer to process, H3 (GRASP) is consistently worst and therefore last in the rankings for both hyper-heuristics. Such rankings are consistent throughout the three utility functions.

Table 4.5: Usage count of metaheuristics for the two hyperheuristics (HH1, HH2) using different utility functions.

Instance	Heuristic	HH1			HH2			Mean CPU time per column (s)
		Count per Utility						
		U1	U2	U3	U1	U2	U3	
I1	H1	44	49	42	36	100	58	0.09
	H2	15	19	20	9	10	11	0.01
	H3	14	19	19	8	10	8	102.55
I2	H1	19	17	28	21	17	20	0.05
	H2	14	11	20	15	13	12	0.01
	H3	10	11	16	14	10	12	193.96
I3	H1	40	38	36	30	44	41	0.11
	H2	16	18	15	11	11	12	0.01
	H3	13	18	15	9	10	12	127.10
I4	H1	5	5	10	10	5	10	0.06
	H2	3	3	4	3	3	3	0.01
	H3	2	2	4	3	2	2	51.58
I5	H1	29	29	22	28	31	29	0.04
	H2	15	21	17	11	14	13	0.01
	H3	12	16	15	11	12	12	102.16

Tables 4.6 and 4.7 show the computational results for all instances using hyper-heuristics HH1 and HH2, respectively, with the three different utilities. Included in the

tables are, the number of iterations, the number of columns generated with negative reduced cost, the percentage deviation from the optimal integer solution, and the total computational time. For every instance, the fastest utility is highlighted in green. The percentage deviation values are calculated, after the last iteration of the hybrid solution procedure, using the difference between the solution provided by the LP relaxation and the integer solution of the master problem. Due to the diving heuristic, close-to-optimal, and even optimal, integer solutions can be observed for three out of the four instances solved without exceeding the maximum number of iterations allowed (I2). Using HH1, all utilities U1 is faster for 4 (out of 5) instances. Using HH2, U1 is faster for 2 (out of 5) instances, while U2 and U3 for 1 and 2 of the remaining instances, respectively. Moreover, HH1 shows significantly higher computational times; for example, for the largest instance (I2), HH1 provides a solution in 67 minutes versus 40 minutes using HH2; HH2 is, on average, 22% faster than HH1. Also, it appears that HH1 does not provide significant improvements in the solutions with respect to HH2 (as seen by the percentage deviation).

For even faster results, we propose the following set up. As seen in Table 4.5, the bottleneck when generating columns comes from H3. Therefore, we propose a modified hyper-heuristic based on HH2 with utility U1 (as suggested by our previous results and the original authors, Özcan et al., 2010) and a reduced set of heuristics for performance, $\mathcal{H}_k = \{H1, H2\}$ for all k subproblems. We refer to this method as mHH2.

Table 4.6: Computational results of the hybrid solution procedure using hyperheuristic HH1.

Instance	Utility	Iterations	Columns	% Deviation	CPU Time (min)
I1	U1	12	49	0.00	43.13
	U2	23	86	0.00	102.22
	U3	14	59	0.00	74.41
I2	U1	50	220	-	37.03
	U2	50	191	-	85.52
	U3	50	192	-	94.28
I3	U1	11	55	0.00	33.56
	U2	17	53	17.12	48.15
	U3	13	57	22.48	48.92
I4	U1	6	7	0.00	4.35
	U2	6	7	0.00	4.43
	U3	6	6	0.00	4.49
I5	U1	7	27	0.00	30.28
	U2	7	29	0.00	29.98
	U3	7	27	0.00	31.27

Table 4.7: Computational results of the hybrid solution procedure using hyperheuristic HH2.

Instance	Utility	Iterations	Columns	% Deviation	CPU Time (min)
I1	U1	18	55	0.00	31.46
	U2	18	50	4.31	36.71
	U3	24	74	0.00	37.76
I2	U1	50	194	-	15.06
	U2	50	224	-	14.25
	U3	50	156	-	16.37
I3	U1	15	62	17.90	23.85
	U2	10	45	21.09	23.77
	U3	13	57	25.31	20.82
I4	U1	7	9	0.00	5.35
	U2	6	7	0.00	4.30
	U3	5	6	0.00	3.62
I5	U1	9	38	0.36	25.61
	U2	8	28	0.00	26.85
	U3	11	42	0.00	32.59

4.5.2 Recovery

Using the more computationally efficient setup, mHH2, we now study the effects of of two types of disruption, aircraft delay and cancellation, and airport closure (temporary and permanent). The number of affected aircraft ranges from 1 to 5 and the duration of a delay from 24 to 144 hours (in steps of 24 hours). For airport closure we consider the closure of the hub airport (varies per instance) for the same range of duration as the aircraft (24 to 144 hours and full recovery period).

Figures 4.4 to 4.8 show the results for each of the five instances considered. The figures on the left show the effect of airport disruption for the ranging set of durations. The figures on the right show the aircraft disruption cases for 1 to 5 aircraft for the different durations. In both of these figures, against the left-hand y -axis, we plot the percentage of the original flights that were disrupted in the resulting schedule. This includes cancelled and delayed flights. Hence, this axis is proportional to the recovery cost, which does not take individual passenger delays and cancellations into account. Against the right-hand y -axis, we have, in a red dashed line, the total and average CPU time for the left and right figures, respectively.

The average computational times for the different duration of airport closures range from just over 30 seconds for the smallest instance (I4) and around 10 minutes for the largest instance (I2). Results suggest, that airport disruptions, apart from being more disruptive also seem more predictable. In three out of five of the instances considered, a full airport closure (“cancel”) is not preferred over a temporary closure. For instances I2 and I5, Figures 4.5 and 4.8 (left), the closure of the airport for a short

duration has a lesser disruptive effect than a longer closure. However, this is most likely due to the fact that the recovery cost does not include factors like passenger delay and other additional costs that arise from a full airport closure. Interestingly, the turning point for the disruption curves seems to occur at after half of the recovery horizon (5 days and 3.5 days).

For aircraft disruptions, we see reasonable computational times, ranging from a maximum of 22 minutes for the largest instance (I2), to less than 30 seconds for the smallest instance (I4). It is not clear, a priori, the best alternative to minimise disruption for each case. There appears to be a connection between the highly volatile aircraft disruptions and the convex airport disruption curves.¹ For instance, Figure 4.4 (right) for instance I1, shows a well-behaved airport disruption curve, and a more intuitive aircraft disruption evolution. Displaying percentage of flights disrupted mostly ordered with the number of aircraft disrupted. On the other hand, instances I2 and I5, in Figures 4.5 and 4.8 (right), show a convex disruption curve and a highly variable aircraft disruption. In these two cases, it seems that for fewer aircraft, less than half of the aircraft available (3 for I2 and 4 for I5), cancellation may be preferred over a range of delay to minimise flight disruption.

¹It is worth noting that other works have also observed comparable fluctuations (Maher, 2015).



Figure 4.4: Effect of two types of disruption on instance I1, airport (left) and aircraft (right).

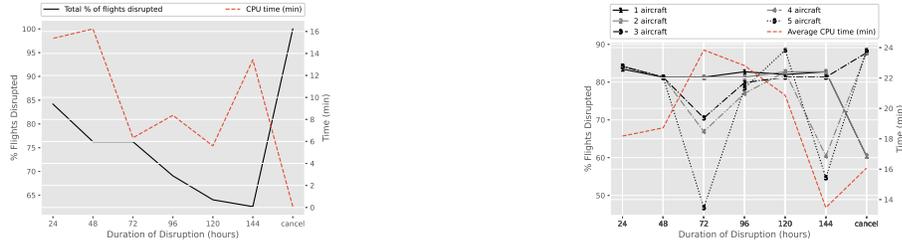


Figure 4.5: Effect of two types of disruption on instance I2, airport (left) and aircraft (right).



Figure 4.6: Effect of two types of disruption on instance I3, airport (left) and aircraft (right).



Figure 4.7: Effect of two types of disruption on instance I4, airport (left) and aircraft (right).



Figure 4.8: Effect of two types of disruption on instance I5, airport (left) and aircraft (right).

4.6 Conclusions

We have solved the aircraft recovery problem considering operational restrictions. Previous studies that tackle a similar problem pose restrictive assumptions on the operational constraints. By using our hybrid solution procedure we are able to account for generic and multiple operational restrictions. Such framework uses column generation with subproblems modelled as shortest path problems with resource constraints (SPPRC). Furthermore, to improve computational efficiency and the quality of the solutions, we implement different hyperheuristic algorithms. Such algorithms learn and select varying metaheuristic or exact algorithms, depending on their past performance, to employ for the generation of columns with negative reduced cost. Using the exact bidirectional labelling algorithm with dynamic halfway point, we produce optimal solutions for the LP relaxation of the aircraft recovery problem. Furthermore, to preserve computational efficiency, a diving method with limited discrepancy search, from Sadykov, Vanderbeck, Pessoa, Tahiri and Uchoa (2019), is applied to obtain close-to-optimal integer solutions to the original IP. Therefore, this paper bridges between several disjoint areas in the literature to provide a valuable contribution. The implementations of the algorithms used for the SPPRC are freely available as a Python package Torres Sanchez (2020).

The hybrid solution procedure was tested using six real-world instances of different sizes. Results show that our approach is not only efficient, solving all instances in reasonable computational time, but it also produces better solutions than the exact algorithm alone and than a random metaheuristic selection process. For the instances

studied, our Tabu search metaheuristic performed consistently better (i.e. providing more negative reduced cost columns) than our Greedy Elimination metaheuristic and an adapted GRASP metaheuristic. Also, the quality of the recovery solution seems to increase with the size of the network. The use of a fairer hyper-heuristic, one that brings a fairer exploration of the metaheuristics, does not lead to a higher quality of recovery solutions. Furthermore, we suggest a computational setup for the hybrid solution procedure that produces faster solutions. Using this set up, we were able to solve various disruption scenarios in reasonable computational times. Particularly, we study the effect of disruptions on aircraft (delay and cancellation) and airports (temporary and full closure). The minimisation of flight disruption for these scenarios reveals complex patterns across different instances and disruption types. Hence, this framework equips decision makers with an efficient learning-based tool that can handle various network sizes and presents insightful solutions to the aircraft recovery problem subject to a range of different disruptions.

Some limitations are worth noting. We have used a reinforcement learning selection hyper-heuristic in this work, nevertheless, the implementation of some of the more advanced forms of hyper-heuristics would be an interesting extension. Additionally, including more SPPRC algorithms would also enrich the hyper-heuristic selection. Even though not typically considered in the literature, the inclusion of ferrying (operating empty flights) brings more flexibility when generating schedules and would, therefore, be a valuable development. With the graph representation employed in this paper time-space network (TSN), ferrying (flying empty aircraft) could be accounted

for by either considering each TSN as a multigraph with a new edge alongside every flight delay copy or in a similar fashion as Thengvall et al. (2001). Provided with ferrying costs, one may use these to set an appropriate edge weight. Nonetheless, the multigraph modelling requires updating the solution algorithms to perform on these types of graphs. Lastly, to preserve computational performance we employed a diving heuristic to provide close-to-optimal integer solutions. However, a branch-and-price framework is required if one aims to produce optimal integer solutions.

Chapter 5

Thesis Conclusions, Limitations and Further Work

This thesis has studied the recent developments in airline fleet maintenance scheduling and other areas, such as aircraft recovery, and presented several contributions. Particularly, we have developed new models and efficient solution methods, and implemented cutting-edge algorithms; providing practical solutions with their companion software.

The first contribution of this thesis focuses on the modelling of, among other factors, the prognostic states of the aircraft and the resource allocation at different workshops, while considering tail reassignments. This was the subject of Chapter 3. The second contribution solves the aircraft recovery problem, generating schedules while taking airline operational restrictions into consideration and minimising disruptions and operational costs. The operational restrictions include aircraft, maintenance, crew duty, and passenger delay. This work formed Chapter 4.

In **Chapter 3**, we resolve the pre-operational conflicts between a proposed flight schedule and maintenance decisions. Although optimisation models that study aircraft maintenance have been developed, certain aspects have not been fully addressed. Such models mainly produce cyclic maintenance schedules for short-term mainte-

nance; and albeit considering a single airline, no individual workshop restrictions or legal remaining flying hours; they tend to be inefficient even for small-medium data sets. In this paper we present a framework that addresses these gaps. The proposed framework deals with the requirements introduced by a 30-day planning horizon, multiple airlines and workshops, individual legal remaining flying hours, and tight resource availabilities. Producing adaptive maintenance schedules for large data sets while remaining computationally efficient. Such framework involves a series of preprocessing steps, two multi-objective mixed integer linear programming (MMILP) formulations, and an iterative algorithm. First, the preprocessing steps allow us to extract maintenance opportunities (MOPs), long turnaround times at maintenance workshops from flight schedules. These MOPs enable us to model the problem using time intervals, as opposed to a traditional time discretisation, which significantly aids the efficiency of our framework. Then the MMILP formulations model the airline fleet maintenance scheduling problem with tail assignment considerations. We minimise six lexicographically ordered objectives which include the number of regulations violations, maximum resource level, number of tail reassignments, number of maintenance interventions, overall resource usage, and the amount of maintenance required by each aircraft at the end of the planning horizon (maximise aircraft “health”). Lastly, the iterative algorithm, ensures a balance between computational efficiency and good quality solutions. Computational tests reveal that our solutions are near-optimal. More importantly, tests highlight the importance of collaborations among airlines and maintenance providers in order to attain a balance between the joint commercial interests and regulations imposed by aviation authorities.

Some limitations of this chapter are worth mentioning. The termination criteria for the algorithm does not guarantee an optimal solution. When compared with a traditional discretisation with a small time step, we see that our solution is better and considerably faster. Thus, suggesting that our solution is near-optimal; however, the proof is likely to be a non-trivial task. Additionally, the chapter does not consider the complications that may arise from some of the long-term maintenance types. For instance, the inclusion of inventory control. Further work includes the implementation of a rolling horizon to lengthen the planning horizon and the exploration of clustering methods to model larger instances with a higher frequency of multi-workshop interaction.

In **Chapter 4**, we solve the aircraft recovery problem using a new hybrid solution procedure. This approach combines the use of column generation with reinforcement learning selection hyper-heuristics to solve the aircraft recovery problem with multiple operational considerations. Specifically, we consider operational restrictions for aircraft, maintenance, crew duty, and passenger delay. Moreover, our generic framework can be easily extended for further considerations. This is due to the modelling of the column generation subproblems as shortest path problems with resource constraints (SPPRC). Hence, allowing each restriction to be modelled as a separate resource with no restriction on the number of resources. Given that this can heavily condition the solution times for the subproblems, we propose two new, and implement two recent metaheuristic algorithms to solve the SPPRC. Additionally, we implement a bidirectional labelling algorithm with a dynamic halfway point to solve the subproblems

exactly. The implementation of these algorithms was released as a Python package (Torres Sanchez, 2020). Computational tests reveal that this approach is computationally efficient for reasonably sized real-world test instances.

There are a number of limitations of this work. The hybrid solution procedure finds close-to-optimal integer solutions for the aircraft recovery problem, by employing a diving heuristic as part of the hybrid solution procedure. However, the initial problem is an IP, therefore, to obtain optimal integer solutions the implementation of a branch-and-price scheme is required. Some authors have managed to implement a branch-and-price scheme with SPPRC subproblems, Feillet et al. (2007), for example. As part of the solution procedure a reinforcement learning selection hyper-heuristic is employed, nonetheless, the exploration of other hyper-heuristics would be an attractive research direction.

Final Remarks The two main chapters of this thesis have four common features.

The modelling of the problems was done using efficient mathematical formulations. In Chapter 3, we introduced an interval-based formulation; while in Chapter 4, we employed a traditional set covering formulation.

Varied solution algorithms were developed and implemented. In Chapter 3, we developed a new problem-specific iterative algorithm that exploits the efficiency provided by the interval-based formulations; in Chapter 4, we developed a hybrid solution procedure that consists on a combination of column generation with hyper-heuristics.

Real-world test instances were solved in reasonable computational time. In Chapter 3, we solved multi-workshop multi-airline instances, the largest of which contained

8 workshops, 16000 flights and 429 aircraft and was solved in under 2 hours. In Chapter 4, we solved multiple test instances the largest of which contained 658 flights and 28 aircraft and was solved in under 70 minutes.

Solution algorithms produced near-optimal solutions. In Chapter 3, test results, with a comparison against a solution obtained using a small traditional time discretisation, suggest that our solution is close to the optimal. In Chapter 4, the implementation of an exact algorithm in the column generation framework ensures that the optimal solution to the LP relaxation is found; while the diving heuristic, provides near-optimal integer solution to the original IP.

Appendix A

Appendix for Chapter 3

A.1 Extension: Airline Maintenance Scheduling with Flight Re-Scheduling

An alternative for the resolution of violations not included in the paper, is one which, instead of tail reassignments, allows flight delays. Recall that Model 3.3.1 identifies maintenance regulation violations. To deal with these, we can allow for some flights to be delayed. From the airlines' perspective, if planning is far enough ahead (which pre-operational planning is), they can notify their staff and customers at no extra cost. This opens the possibility to extend a MOP by the precise amount of time that allows a maintenance intervention to be performed. Our aim is, therefore, to delay the smallest number of flights by the least amount of time so as to avoid knock-on delays.

For Model 3.3.1 to allow flight delays, we simply require an extra variable and the modification of constraints 3.3.10. Let p_k^f be a continuous variable that represents the duration of a delay for flight f and aircraft k . We can limit the maximum single flight delay to be Δ_p . We assume that the delays introduced, if kept short, can be absorbed by the (large enough) turnaround time (TAT) between two MOPs.

The additional constraints to extend Model 3.3.1, are as follows,

Minimum Time

$$\sum_{i' \in MOP_k^j} (et^{i'} - st^{i'})m_{kc}^{i'} + p_k^f \geq \Delta_{t(k),c}m_{kc}^i \quad \forall k, c, j, i \in MOP_k^j, f \in \mathcal{F}_{dep}^i; \quad (\text{A.1.1})$$

Variables

$$0 \leq p_k^f \leq \Delta_p \quad \forall f, k \in \mathcal{K}^f; \quad (\text{A.1.2})$$

Constraints A.1.1 account for the additional time that can be allocated to maintenance if the flight in question is delayed. Constraints A.1.2 bound the delay variable. Additionally, to minimise delay in Model 3.3.1, we simply add a third objective, with the total delay introduced.

A.2 Supplementary Proofs

Proposition A.2.1. *For a given interval i and aircraft k , the following inequality holds,*

$$\sup\{MOP_k^{j-1}\} < i \leq \sup\{MOP_k^j\},$$

for some MOP_k^j provided that $j \neq 1$, $i > \sup\{MOP_k^1\}$ and $i \leq \sup\{MOP_k^J\}$ ($J = |MOP_k|$). In other words, provided that the interval under consideration starts after the end of the first MOP and before the end of the last MOP.

Proof. For a given i and aircraft k , either, $i \in \mathcal{I}_k$ or $i \notin \mathcal{I}_k$.

In the first case, by definition $i \in MOP_k^j$ for some j , hence, clearly, $i \leq \sup\{MOP_k^j\}$.

Also, by definition, intervals in MOPs are disjoint, $i \notin MOP_k^{j-1}$, thus $i > \sup\{MOP_k^{j-1}\}$.

Therefore, the inequality holds.

Otherwise, we have,

$$i \notin \mathcal{I}_k \wedge i \leq \sup\{MOP_k^J\} \implies \exists j = \max\{j: j \in MOP_k, j \leq J, i < \inf\{MOP_k^j\}\} .$$

Hence, $i \leq \sup\{MOP_k^j\}$, and, therefore, $i > \sup\{MOP_k^{j-1}\}$. This completes the proof. \square

Proposition A.2.2. *Transitivity constraints 3.3.11 – 3.3.13 in Model 3.3.2, ensure non-preemption for all MOPs.*

Proof. We want to prove the following relationship,

$$\text{Transitivity constraints} \implies \text{Non-preemption},$$

or, equivalently,

$$\text{Preemption} \implies \text{Transitivity constraints do not hold.}$$

Suppose that, maintenance begins at an interval $i \in MOP_k^j$, or,

$$m_{kc}^{i-1} = 0 \wedge m_{kc}^i = 1 \text{ for some } i, i-1 \in MOP_k^j .$$

Using constraints 3.3.12, we have that $z_{kc}^i = 1$. Now, assume there is a preemption, that is,

$$m_{kc}^{i'-1} = 0 \wedge m_{kc}^{i'} = 1 \text{ for some } i' \in MOP_k^j, i' > i .$$

By constraints 3.3.12, we have that $z_{kc}^{i'} = 1$. However, given constraints 3.3.13 and $z_{kc}^i = 1$, we have a contradiction. Therefore, we must have

$$m_{kc}^{i'} - m_{kc}^{i'-1} \leq 0 \quad \forall i' \in MOP_k^j, \quad i' > i.$$

Hence, within an MOP, once maintenance has started, we can either continue, in which case $m_{kc}^{i'} - m_{kc}^{i'-1} = 0$, or end it, where $m_{kc}^{i'} - m_{kc}^{i'-1} = -1$. We cannot start another maintenance intervention within the same MOP.

For the boundary condition, if i is the first interval in the MOP, $i = \inf\{MOP_k^j\}$, then by constraints 3.3.11 and 3.3.13, the claim still holds. This completes the proof.

□

A.3 Interval Generation

The following framework allows us to generate the intervals used in the formulation. Actually, aircraft intervals, \mathcal{I}_k , used in the formulation are a specific case of interval as defined here. Let us begin by visualising and using the flight routes and maintenance locations; for this purpose we construct a multigraph. A multigraph differs from an ordinary graph in that two vertices are allowed to have more than one edge between them. Moreover, we consider a directed and weighted multigraph which has similar properties to its counterpart. Let the flight schedule be represented such a multigraph $M = (V, E)$; $|V| = n$, $|E| = m$. The vertex set $V = \{s_1, \dots, s_n\}$ contains both the airports and the maintenance locations. The edge set E contains different flights.

The weight function $w = (w_1, w_2)$ contains information about departure time w_1 and flight times w_2 , is attributed to each edge. ($w_k : E \rightarrow \mathbb{R}, k = 1, 2$).

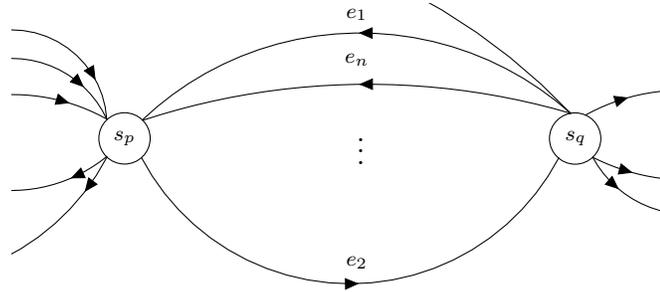


Figure A.1: Example of edge labelling for flights between airports s_p and s_q .

The weight function allows for an edge labelling procedure that orders the edges by departure time,

$$w_1(e_1) \leq w_1(e_2) \leq \dots \leq w_1(e_N) \leq \dots \leq w_1(e_M).$$

Clearly, this edge labelling, which we denote as \mathcal{L}_E , is unique.

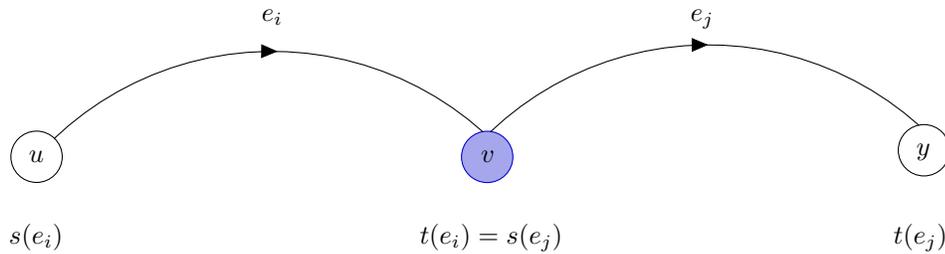


Figure A.2: Incident edges with source and sink functions.

Since we do not need to know exactly where each aircraft is at every period of time, but rather, how its engines are being used and when its next maintenance opportunity might be. We can narrow down the network to remove all the unnecessary airports and group them into a supersource and a supersink. Let $S \subseteq V$ be a set of

locations which can perform at least one type of maintenance check, with

$$E_S = \{e : e \in E, t(e) = s \vee s(e) = s \text{ for some } s \in S\},$$

let $|E_S| = Q$ and $\mathcal{L}_{E_S} \subseteq \mathcal{L}_E$ be the corresponding labelling alphabet. Where $s(\cdot)$ and $t(\cdot)$ are the ordinary source and sink functions. For notation purposes, we map the labelling alphabet \mathcal{L}_{E_S} to a set of integers $\mathcal{L}_Q = \{1, \dots, Q\}$ with $e_i \in \mathcal{L}_{E_S} \rightarrow \tilde{i} \in \mathcal{L}_Q$.

We can construct a refined incidence matrix “edge-adjacency” (Zoethout, 2015), \mathcal{Q} , as follows

$$q_{(\tilde{i}, \tilde{j})} = \begin{cases} 1, & \text{If } i < j \wedge t(e_i) = s(e_j) \wedge w_1(e_i) + w_2(e_i) + TAT < w_1(e_j) \\ 0, & \text{Otherwise} \end{cases}$$

Therefore, \mathcal{Q} is a $Q \times Q$ strictly upper triangular matrix containing nonzero elements only if the turn-around-time (TAT) (time from arrival to the next departure) is large enough. That is, if flight j 's departure is at a later time than i 's ($i < j$) and the arrival of flight i plus some TAT is less than the departure time of flight j ; then we may consider employing this time window for maintenance. Hence, we have identified a maintenance opportunity. The values of the TAT can make \mathcal{Q} represent either a more relaxed feasible flight connections or a more restrictive (and perhaps more robust) feasible intervals for maintenance checks.

We now show that N is linear in Q , the number of flights strictly between maintenance sites. Since N determines the number of variables, as they are indexed by intervals, this proposition shows that the model is tractable for a reasonable Q .

Proposition A.3.1. *An upper bound for N , the number of intervals, is*

$$0 < N \leq \min\{\Delta^+, \Delta^-\}Q \ll \left\lfloor \frac{Q^2 - Q}{2} \right\rfloor$$

where Δ denotes the maximum indegree (+) and outdegree (−) of the vertices in \mathcal{S} .

Proof. For every edge in $e_i \in E_S$ we can easily identify the outdegree of its sink vertex, that is $\text{deg}_{\text{out}}(t(e_i))$. Each row \tilde{i} of matrix \mathcal{Q} represents the possible connections for flight e_i . For each of these, the maximum number of possible connections is the outdegree of the sink vertex. For clarity, we drop the tilde notation. By the edge labelling, a connection between e_i and e_j is not possible if $j \leq i$. Thus, we can reduce the initial number of connections for each flight e_i to

$$\sum_{j=i+1}^Q q_{(i,j)} \leq \text{deg}_{\text{out}}(t(e_i)) - |E_{t(e_i)}|. \quad (\text{A.3.1})$$

Where $E_{t(e_i)} = \{e_j : e_j \in E_S, j > i, t(e_i) = s(e_j)\}$. For completeness, we also need to count the outdegree of the edges ingoing and outgoing to vertices in the set of maintenance locations \mathcal{S} we can define a super source S and a super sink T . For every $e \in E_S$ if $s(e) \notin \mathcal{S}$ then $s(e) = S$, if $t(e) \notin \mathcal{S}$ then $t(e) = T$; let $\mathcal{S}' = \mathcal{S} \cup \{S, T\}$.

The upper bound comes from considering the sum of all the nonzero elements in the part of the upper triangular matrix \mathcal{Q} , if every single element was 1, then the total number of elements would be $\left\lfloor \frac{Q^2 - Q}{2} \right\rfloor$. However, this is not the case and we can sum equation A.3.1 to represent the total number of all the nonzero elements in the

matrix \mathcal{Q} i.e. the size of \mathcal{P} . Thus,

$$\sum_{i=1}^Q \sum_{j=i+1}^Q q_{(i,j)} = N \leq \sum_{i=1}^Q [deg_{out}(t(e_i)) - |E_{t(e_i)}|] = \sum_{i=1}^Q deg_{out}(t(e_i)) - \sum_{i=1}^Q |E_{t(e_i)}| \quad (\text{A.3.2})$$

Now, we can expand the first sum in equation A.3.2 of the RHS directly in terms of the vertices as opposed to the sink of the edges. That is,

$$\sum_{i=1}^Q deg_{out}(t(e_i)) = \sum_{s \in \mathcal{S}'} deg_{out}(s) deg_{in}(s)$$

now, by the first theorem of multigraph theory. equivalent to hand-shaking lemma (Van Steen, 2010), and by Abel's inequality (Dragomir et al., 1998), we get,

$$\sum_{s \in \mathcal{S}'} deg_{out}(s) deg_{in}(s) \leq \min\{\Delta^+, \Delta^-\} Q$$

by commutativity in the product of the non-negative degree functions.

For the second term in equation A.3.2, we have $\max\{|E_{t(e_i)}|\} = deg_{out}(t(e_i)) \leq \Delta^+$ and $\min\{|E_{t(e_i)}|\} = 0$. The result follows. \square

Appendix B

Appendix for Chapter 4

B.1 Functions Employed in Metaheuristic Algorithms

B.1.1 Algorithm 5

The functions employed in Algorithms 5 is defined as,

```
function GETDIRECTION(F, B)
  if F  $\neq$   $\emptyset$  and B =  $\emptyset$  then
    direction = forward
  else if F =  $\emptyset$  and B  $\neq$   $\emptyset$  then
    direction = backward
  else if F  $\neq$   $\emptyset$  and B  $\neq$   $\emptyset$  then
    Randomly set direction to be either forward or backward.
  else
    direction = None
  end if
end function

function CHECKDOMINANCE(Q, direction, S)
   $\triangleright$  Check dominance for a given path, direction and path extension set
  for Q' comparable paths in S do
    if Q dominates Q' then
```

```

        Remove Q' and all of its extensions from S
    else if Q' dominates Q then
        Remove Q and all of its extensions from S
        Break
    end if
end for
if no break then ▷ Input label dominates all labels in input list
    if direction = forward then
        Add Q to FB
    else
        Add Q to BB
    end if
end if
return S
end function

function JOINLABELS(FB, BB)
    for fwd in FB do
        for bwd in BB do
            Merge paths fwd and bwd.
            if merged path is valid (source-sink path), resource feasible, and satisfies
halfway check then
                Save merged path
            end if
        end for
    end for
end function

```

B.1.2 Algorithms 6 and 7

The shortest path function employed in Algorithms 6 and 7 is defined as,

```

function SHORTESTPATH( $i, j, G$ )
    Apply A* search algorithm to find the shortest path between nodes  $i$  and  $j$ , in
     $G$ .
end function

```

The function above uses an A* search algorithm (Hart et al., 1968) to find a shortest path between two nodes. However, this relies on the assumption that the input graph has no negative cycles. Alternatively, if one wishes to relax this assumption,

(as it happens with many cases in column generation), other algorithms can be used in this function. For instance, in the most recent version of `cspy` (Torres Sanchez, 2020), we use a standard algorithm (Yen, 1970) to compute all k -shortest simple paths with no edge weights. Once these have been obtained, the real costs of the paths are evaluated and the one with the smallest cost is chosen.

The functions employed in Algorithm 6 are as follows,

```

function UPDATEPATH(neighbour,  $p$ )
  if iter= 0 then
    path=  $p$ 
    if neighbour in path then
      Concatenate elements in path, up until neighbour, with  $p$ 
    else
      Concatenate elements in path, up until an element node such that an
      edge (node, neighbour) exists in  $G$ , with  $p$ 
    end if
  end if
end function

function GETNEIGHBOUR(edgesToCheck, neighbourhood,  $g$ )
  if edgesToCheck is not empty then ▷ There still remains edges to try
    if  $g$  in edgesToCheck then ▷ If edge has not already been checked
      currentEdge =  $g$ 
    else if  $h(g)$  = Source then ▷ If the source has been reached
      stop = True ▷ Terminate the algorithm
      return
    else ▷ Otherwise try an edge from the predecessor array of edges
      currentEdge, neighbourhood = GETNEXTNEIGHBOURINGEDGE(neighbourhood,
      tabu)
    end if
  else
    stop= True ▷ Stop if all edges have been removed
    return
  end if
  Remove currentEdge from edgesToCheck and set its weight to  $M$  (input large
  number)
  return currentEdge, edgesToCheck, neighbourhood
end function

function GETNEXTNEIGHBOURINGEDGE(neighbourhood,  $l$ )

```

```

if neighbourhood is empty then
    Update neighbourhood with all edges  $e'$  in  $A$  if  $t(e') = h(l)$ 
    Sort neighbourhood in increasing order by edge cost
end if
Let edge be the last element in neighbourhood and remove it from neighbourhood
return edge, neighbourhood
end function

```

B.1.3 Algorithm 8

The functions employed in Algorithm 8 are as follows,

```

function CONSTRUCTSOLUTION()
    candidates, RCL = [] ▷ Initialise local arrays
    solution = SAMPLE( $V$ ) ▷ Initialise solution array by sampling a random node
    while |solution| <  $|V|$  do ▷ While solution does not contain all nodes
        Populate candidates array using all nodes not in solution
        candidate = GETRCL(candidates, solution)
        solution = solution  $\cup$  candidate
    end while
    return solution
end function

```

```

function GETRCL(candidates, solution)
    ▷ Constructs a restricted candidate list (RCL) and returns one candidate
    costs = [] ▷ Initialise local arrays
    Populate costs with the cost between each node in candidates and the last
    node in solution
    for node  $n$  in candidates do
        if costs[ $n$ ]  $\leq$  min(costs) +  $\alpha$ (max(costs) – min(costs)) then
            Add node  $n$  to RCL
        end if
    end for
    return random node in RCL
end function

```

```

function LOCALSEARCH(solution)
    ▷ Conducts local search to return a valid path
    localiter = 0 ▷ Initialise local iteration counter
    while localiter < maxiterLocal do
        Find a path in  $G$  using random sized samples of permutations of the nodes
        in solution.
        Save the path as candidate
    end while

```

```

    if  $c(\text{candidate}) < c(\text{solution})$  and candidate is resource feasible then
        solution = candidate
    end if
    localiter = localiter + 1           ▷ Increment local iteration counter
end while
return solution
end function

```

B.2 Parameters in the Computational Tests

Table B.1 reveals the precise aircraft makes and types for each test instance, their respective range, standard total operational costs (as given by FAA, 2013) and operational costs under delay (as given by Cook and Tanner, 2015). For illustration and recalling the notation from Section 4.3.3, for a given subproblem k , with the make of aircraft k being DH8D, then, the standard total operating cost for a scheduled flight is $c_k^{\text{st}} = \$922$ (around €823 at the current exchange rate), the cost for non-scheduled ground connections is $c_k^{\text{gw}} = €540$, and, the operating cost under delay for flight delay copies is $c_k^{\text{od}} = €1630$.

Table B.2 summarises the parameters employed in the computational tests. First, the cancellation cost, captured in the costs of the initial schedules in the column generation algorithm (in the notation of Section 4.3.3, c^s where $s \in \mathcal{S}^0$), were given by Palpant et al. (2009). The minimum duration of a crew duty break, `breakMin`, is given by Federal Aviation Administration (2009). The minimum duration for a maintenance intervention, `maintMin`, is given by Torres Sanchez et al. (n.d.). The remaining parameters concern to the resource limits introduced in Section 4.3.3. These ensure that the appropriate regulations are met. Please note that the values for

maintenance (FH) and crew duty rules (CD1, CD2) were given by Cook and Tanner (2008) and Maher (2015), respectively.

Table B.1: Different aircraft types in test instances with their respective operating costs.

Aircraft Make	Aircraft Type (Range)	Inst.	Operating Costs (per hour)		
			Standard (c_k^{st})	Ground (c_k^{gw})	Delayed Flight (c_k^{od})
DH8D	Turboprop (Short-haul)	I1	\$922	€540	€1630
JS32	Turboprop (Short-haul)	I2	\$2887	€540	€1630
A319	Narrow-body (Short/medium-haul)	I1, I4	\$9734	€810	€3420
A320	Narrow-body (Short/medium-haul)	I5	\$9734	€900	€3490
B738	Narrow-body (Short/medium-haul)	I6	\$10430	€1010	€3650
B752	Narrow-body (Short/medium-haul)	I6	\$10430	€720	€4210
A359	Wide-body (Long-haul)	I3, I4	\$13912	€1050	€4130
B77L	Wide-body (Long-haul)	I6	\$13912	€1310	€6230

Table B.2: Parameters for computational tests.

Parameter	Representation	Value
c^s ($s \in \mathcal{S}^0$)	Flight cancellation cost (per hour)	€20000
breakMin	Minimum crew duty break	3 hours
maintMin	Minimum maintenance time	5 hours
FH	Maximum flying hours between maintenance (A check)	600 flying hours
CD1	Maximum crew shift duration (Duty rule 1)	13 hours
CD2	Maximum crew flying hours without rest (Duty rule 2)	8 hours

B.3 `cspy`: A Python package with a collection of algorithms for the (Resource) Constrained Shortest Path problem

B.3.1 Introduction

When solving the shortest path problem and considering multiple operational restrictions, one may resort to the (resource) constrained shortest path (CSP) problem. It consists, as its name suggests, in finding, among all paths, the shortest path from source to sink nodes that satisfies a set of constraints for a defined set of resources. Such set of resources and the way they evolve throughout the path are user defined and controlled. This allows the modelling of a wide variety of problems including: the vehicle routing problem with time windows, the technician routing and scheduling problem, the capacitated arc-routing problem, on-demand transportation systems, and, airport ground movement (Desrochers and Soumis, 1988; Feillet et al., 2004; Irnich and Villeneuve, 2006; Righini and Salani, 2008; Bode and Irnich, 2014; Chen et al., 2016; Garaix et al., 2010; Tilk et al., 2017; Zamorano and Stolletz, 2017).

`cspy` is a Python package that allows you to solve instances of the CSP problem using up to eight different algorithms.

`cspy` is of interest to the operational research community and others that wish to solve an instance of the CSP problem.

B.3.2 Algorithms

Even though the CSP problem is \mathcal{NP} -hard (Garey and Johnson, 1990), several algorithms have been developed to solve it. The most common algorithms are dynamic programming labelling algorithms. Irnich and Desaulniers (2005) presented an exact algorithm based on DP, the monodirectional forward labelling algorithm, based on the pioneering work by Desrochers and Soumis (1988).

Advanced and efficient algorithms have been developed since. Boland et al. (2006) published a state augmenting algorithm that uses a monodirectional labelling algorithm to find an elementary path (one without repeating nodes). Such algorithm has been implemented by Weyens (2018).

Righini and Salani (2006) introduced a bidirectional labelling algorithm for the SPPRC. The bidirectional algorithm is an extension of the monodirectional algorithm that supports search from both ends of the graph, hence, reducing the computational efforts. More recently, Tilk et al. (2017) developed a bidirectional labelling algorithm with dynamic halfway point. The bidirectional search is bounded for both directions and these bounds are dynamically updated as the search in either direction advances. The algorithm has shown to be significantly more efficient than monodirectional ones (Gschwind et al., 2018).

Even with some of the most recent algorithms, solving an instance of the SPPRC can be slow, thus, heuristic algorithms have been developed. Local search or meta-heuristics start with a given path and perform a series of moves (edge/node deletion, insertion, or exchange) to obtain another feasible path with lower cost. Some meta-

heuristics developed include, Tabu search (Desaulniers et al., 2008), hybrid particle swarm algorithm (Marinakis et al., 2017), and, greedy randomised adaptive search procedure (GRASP) (Ferone et al., 2019).

`cspy` implements several of these recent exact and metaheuristic algorithms including:

- Bidirectional labeling algorithm with dynamic halfway point (Tilk et al., 2017); which includes the bidirectional labeling algorithm with static halfway point, and the monodirectional forward and backward labeling algorithms;
- Tabu search. Adapted from Desaulniers et al. (2008);
- Greedy elimination procedure;
- Greedy Randomised Adaptive Search Procedure (GRASP). Adapted from Ferone et al. (2019);
- Particle Swarm Optimization with combined Local and Global Expanding Neighborhood Topology (PSOLGENT) (Marinakis et al., 2017).

B.3.3 Features

A key component of a CSP problem is to define a set of resources and a function to apply for these to evolve through the graph. Such functions are typically referred to as resource extension functions (REFs). The simplest REF is an additive one, where every time an edge is traversed a constant unit of a certain resource is consumed. However, custom and more generic REFs can be used.

`cspy` allows for custom and generic REFs to be used. Hence, allowing for a custom consumption of the resources through the graph.

B.3.4 Examples

The package has been used in the following examples:

- `vrpy` : vehicle routing framework which solves different variants of the vehicle routing problem (including capacity constraints and time-windows) using column generation. The framework has been tested on standard vehicle routing instances.
- `cgar` : Complex example using column generation applied to the aircraft recovery problem.
- `jpath` : Simple example showing the necessary graph adaptations and the use of custom resource extension functions.

Bibliography

- Abdelghany, K. F., Abdelghany, A. F. and Ekollu, G. (2008), ‘An integrated decision support tool for airlines schedule recovery during irregular operations’, *European Journal of Operational Research* **185**(2), 825 – 848.
- Ackert, S. (2011), *The Relationship between an Aircraft’s Value and its Maintenance Status*, Technical report, Aircraft Monitor.
- Afshar-Nadjafi, B. (2014), ‘Resource Constrained Project Scheduling Subject to Due Dates: Preemption Permitted with Penalty’, *Advances in Operations Research* .
- Afshar-Nadjafi, B. and Arani, M. (2014), ‘Multimode preemptive resource investment problem subject to due dates for activities: Formulation and solution procedure’, *Advances in Operations Research* .
- Agarwal, Y., Mathur, K. and Salkin, H. M. (1989), ‘A set-partitioning-based exact algorithm for the vehicle routing problem’, *Networks* **19**(7), 731–749.
- Aircraft Analysis & Fleet Planning (2005), *Can the 787 & A350 transform the economics of long-haul services?*, Technical report, Aircraft commerce.

- Allamraju, H. (2014), ‘Flightradar24 Unofficial API: [pyflightdata](#)’. Accessed: 14/08/2019.
- Allen, S., Burke, E. K., Hyde, M. and Kendall, G. (2009), [Evolving reusable 3d packing heuristics with genetic programming](#), in ‘Proceedings of the 11th Annual conference on Genetic and evolutionary computation’, ACM, pp. 931–938.
- Angelelli, E., Bianchessi, N. and Filippi, C. (2014), ‘Optimal interval scheduling with a resource constraint’, *Computers & operations research* **51**, 268–281.
- Argüello, M. F. (1997), [Framework for exact solutions and heuristics for approximate solutions to airlines’ irregular operations control aircraft routing problem.](#), PhD thesis, The University of Texas at Austin.
- Argüello, M. F., Bard, J. F. and Yu, G. (1997), ‘[A GRASP for aircraft routing in response to groundings and delays](#)’, *Journal of Combinatorial Optimization* **1**(3), 211–228.
- Arıkan, U., Gürel, S. and Aktürk, M. S. (2016), ‘[Integrated aircraft and passenger recovery with cruise time controllability](#)’, *Annals of Operations Research* **236**(2), 295–317.
- Arıkan, U., Gürel, S. and Aktürk, M. S. (2017), ‘[Flight network-based approach for integrated airline recovery with cruise speed control](#)’, *Transportation Science* **51**(4), 1259–1287.
- Bae, K. H. (2010), [Integrated airline operations: schedule design, fleet assignment, air-](#)

- craft routing, and crew scheduling, PhD thesis, Faculty of the Virginia Polytechnic Institute and State University.
- Bard, J. F., Yu, G. and Arguello, M. F. (2001), ‘Optimizing aircraft routings in response to groundings and delays’, *IIE Transactions* **33**(10), 931–947.
- Barnhart, C., Boland, N. L., Clarke, L. W., Johnson, E. L., Nemhauser, G. L. and Shenoi, R. G. (1998), ‘Flight String Models for Aircraft Fleeting and Routing’, *Transportation Science* **32**(3), 208–220.
- Barnhart, C., Cohn, A. M., Johnson, E. L., Klabjan, D., Nemhauser, G. L. and Vance, P. H. (2003), *Airline crew scheduling*, in ‘Handbook of transportation science’, Springer, pp. 517–560.
- Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. and Vance, P. H. (1998), ‘Branch-and-price: Column generation for solving huge integer programs’, *Operations research* **46**(3), 316–329.
- Bellman, R. (1958), ‘On a routing problem’, *Quarterly of applied mathematics* **16**(1), 87–90.
- Bianchessi, N., Mansini, R. and Speranza, M. (2014), ‘The distance constrained multiple vehicle traveling purchaser problem’, *European Journal of Operational Research* **235**(1), 73 – 87.
- Bisaillon, S., Cordeau, J.-F., Laporte, G. and Pasin, F. (2011), ‘A large neighbourhood search heuristic for the aircraft and passenger recovery problem’, *4OR* **9**(2), 139–157.

- Bode, C. and Irnich, S. (2014), ‘The shortest-path problem with resource constraints with $(k, 2)$ -loop elimination and its application to the capacitated arc-routing problem’, *European Journal of Operational Research* **238**(2), 415–426.
- Boland, N., Dethridge, J. and Dumitrescu, I. (2006), ‘Accelerated label setting algorithms for the elementary resource constrained shortest path problem’, *Operations Research Letters* **34**(1), 58–68.
- Bratu, S. and Barnhart, C. (2006), ‘Flight operations recovery: New approaches considering passenger recovery’, *Journal of Scheduling* **9**(3), 279–298.
- Brucker, P. and Knust, S. (2012), *Complex Scheduling*, GOR-Publications, 2nd edn, Springer, Berlin, Heidelberg.
- Burke, E. K., De Causmaecker, P., De Maere, G., Mulder, J., Paelinck, M. and Vanden Berghe, G. (2010), ‘A multi-objective approach for robust airline scheduling’, *Comput. Oper. Res.* **37**(5), 822–832.
- Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E. and Qu, R. (2013), ‘Hyper-heuristics: A survey of the state of the art’, *Journal of the Operational Research Society* **64**(12), 1695–1724.
- Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Özcan, E. and Woodward, J. R. (2019), A classification of hyper-heuristic approaches: Revisited, in ‘Handbook of Metaheuristics’, Springer, pp. 453–477.
- Burke, E. K., Kendall, G. and Soubeiga, E. (2003), ‘A tabu-search hyperheuristic for timetabling and rostering’, *Journal of heuristics* **9**(6), 451–470.

- Burke, E. K., McCollum, B., Meisels, A., Petrovic, S. and Qu, R. (2007), ‘A graph-based hyper-heuristic for educational timetabling problems’, *European Journal of Operational Research* **176**(1), 177–192.
- Cacchiani, V. and Salazar-González, J.-J. (2016), ‘Optimal solutions to a real-world integrated airline scheduling problem’, *Transp. Sci.* **51**(1), 250–268.
- Cadarso, L., Vaze, V., Barnhart, C. and Marín, Á. (2016), ‘Integrated airline scheduling: considering competition effects and the entry of the high speed rail’, *Transp. Sci.* **51**(1), 132–154.
- Cardillo, A., Zanin, M., Gómez-Gardeñes, J., Romance, M., García del Amo, A. J. and Boccaletti, S. (2013), ‘Modeling the multi-layer nature of the European Air Transport Network: Resilience and passengers re-scheduling under random failures’, *Eur. Phys. J. Spec. Top.* **215**(1), 23–33.
- Chen, J., Weiszer, M., Stewart, P. and Shabani, M. (2016), ‘Toward a More Realistic, Cost-Effective, and Greener Ground Movement Through Active RoutingPart I: Optimal Speed Profile Generation’, *IEEE Transactions on Intelligent Transportation Systems* **17**(5), 1196–1209.
- Christer, A. H., Wang, W., Sharp, J. and Baker, R. (1998), ‘A case study of modelling preventive maintenance of a production plant using subjective data’, *J. Oper. Res. Soc.* **49**(3), 210–219.
- Chryssolouris, G., Pierce, J. E. and Dicke, K. (1992), ‘A decision-making approach to

- the operation of flexible manufacturing systems’, *Int. J. Flex. Manuf. Syst.* **4**, 309–330.
- Cintra, G., Miyazawa, F. K., Wakabayashi, Y. and Xavier, E. (2008), ‘Algorithms for two-dimensional cutting stock and strip packing problems using dynamic programming and column generation’, *European Journal of Operational Research* **191**(1), 61–85.
- Clarke, L., Johnson, E., Nemhauser, G. and Zhu, Z. (1997), ‘The aircraft rotation problem’, *Annals of Operations Research* **69**(0), 33–46.
- Clausen, J., Larsen, A., Larsen, J. and Rezanova, N. J. (2010), ‘Disruption management in the airline industry– Concepts, models and methods’, *Computers & Operations Research* **37**(5), 809–821.
- Clautiaux, F., Sadykov, R., Vanderbeck, F. and Viaud, Q. (2019), ‘Pattern-based diving heuristics for a two-dimensional guillotine cutting-stock problem with leftovers’, *EURO Journal on Computational Optimization* **7**(3), 265–297.
- Coffman, E. G., Ng, C. T. and Timkovsky, V. G. (2015), ‘How small are shifts required in optimal preemptive schedules?’, *Journal of Scheduling* **18**(2), 155–163.
- Cohn, A. M. and Barnhart, C. (2003), ‘Improving Crew Scheduling by Incorporating Key Maintenance Routing Decisions’, *Operations Research* **51**(3), 387–396.
- Cook, A. and Tanner, G. (2008), Innovative Cooperative Actions of R&D in EURO-CONTROL Programme CARE INO III: Dynamic Cost Indexing: Aircraft mainte-

- nance – marginal delay costs, Technical report, Transport Studies Group, University of Westminster.
- Cook, A. and Tanner, G. (2015), *European airline delay cost reference values: Updated and extended values*, Technical report, University of Westminster.
- Cordeau, J.-F., Stojković, G., Soumis, F. and Desrosiers, J. (2001), ‘Benders Decomposition for Simultaneous Aircraft Routing and Crew Scheduling’, *Transportation Science* **35**(4), 375–388.
- Creemers, S. (2019), ‘The preemptive stochastic resource-constrained project scheduling problem’, *European Journal of Operational Research* **277**(1), 238–247.
- Dantzig, G. B. and Wolfe, P. (1960), ‘Decomposition principle for linear programs’, *Operations Research* **8**(1), 101–111.
- Demasse, S., Artigues, C. and Michelon, P. (2005), ‘Constraint-Propagation-Based Cutting Planes: An Application to the Resource-Constrained Project Scheduling Problem’, *INFORMS Journal on Computing* **17**(1), 52–65.
- Department for BIS (2016), ‘UK Aerospace Maintenance, Repair, Overhaul & Logistics Industry Analysis’, *BIS Research Paper Number 275*.
- DePold, H. R. and Gass, F. D. (1998), *The application of expert systems and neural networks to gas turbine prognostics and diagnostics*, in ‘International Gas Turbine & Aeroengine Congress & Exhibition Stockholm, Sweden’.

- Desaulniers, G., Desrosiers, J., Dumas, Y., Solomon, M. M. and Soumis, F. (1997), ‘Daily aircraft routing and scheduling’, *Management Science* **43**(6), 841–855.
- Desaulniers, G., Lessard, F. and Hadjar, A. (2008), ‘Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows’, *Transportation Science* **42**(3), 387–404.
- Desrochers, M. (1986), La Fabrication d’Horaires de Travail pour les Conducteurs d’Autobus par une Méthode de Génération de Colonnes, PhD thesis, Centre de recherche sur les Transports, Université de Montréal, Montréal, Canada.
- Desrochers, M. and Soumis, F. (1988), ‘A Generalized Permanent Labelling Algorithm For The Shortest Path Problem With Time Windows’, *INFOR: Information Systems and Operational Research* **26**, 191–212.
- Dijkstra, E. W. et al. (1959), ‘A note on two problems in connexion with graphs’, *Numerische mathematik* **1**(1), 269–271.
- Doerr, B., Lissovoi, A., Oliveto, P. S. and Warwicker, J. A. (2018), On the runtime analysis of selection hyper-heuristics with adaptive learning periods, in ‘Proceedings of the Genetic and Evolutionary Computation Conference’, ACM, pp. 1015–1022.
- Dorigo, M. (1992), Optimization, Learning and Natural Algorithms, PhD thesis, Politecnico di Milano.
- Dorigo, M., Birattari, M. and Stutzle, T. (2006), ‘Ant colony optimization’, *IEEE computational intelligence magazine* **1**(4), 28–39.

- Dragomir, S. S., Pearce, C. E. M. and Sunde, J. (1998), ‘Abel-type inequalities, complex numbers and Gauss–Pólya type integral inequalities’, *Mathematical Communications* **3**, 95–101.
- Eggenberg, N., Salani, M. and Bierlaire, M. (2010), ‘Constraint-specific recovery network for solving airline recovery problems’, *Computers & Operations Research* **37**(6), 1014 – 1026.
- Eurocontrol (2011), Study into the impact of the global economic crisis on airframe utilisation., Technical report, Central Office for Delay Analysis, EUROCONTROL. Accessed: 14/08/2019.
- European Parliament and Council of the European Union (2004), ‘Regulation (EC) No 261/2004 of the European Parliament and Council of the European Union’, *Official Journal of the European Union* **46**(1), 1–7.
- FAA (2013), Benefit-Cost Analysis. Section 4: Aircraft operating costs., Technical report, Federal Aviation Administration.
- Federal Aviation Administration (2009), ‘Flightcrew Member Duty and Rest Requirements’, *Federal Aviation Administration (FAA), DOT* .
- Feillet, D., Dejax, P., Gendreau, M. and Gueguen, C. (2004), ‘An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems’, *Networks* **44**(3), 216–229.
- Feillet, D., Gendreau, M. and Rousseau, L.-M. (2007), ‘New Refinements for the

- Solution of Vehicle Routing Problems with Branch and Price', *INFOR: Information Systems and Operational Research* **45**(4), 239–256.
- Feo, T. A. and Bard, J. F. (1989), 'Flight scheduling and maintenance base planning', *Management Science* **35**(12), 1415–1432.
- Feo, T. A. and Resende, M. G. (1995), 'Greedy randomized adaptive search procedures', *Journal of global optimization* **6**(2), 109–133.
- Ferone, D., Festa, P. and Guerriero, F. (2019), 'An efficient exact approach for the constrained shortest path tour problem', *Optimization Methods and Software* **0**(0), 1–20.
- Flightradar24 AB (2018), 'Flightradar24'. Accessed: 14/08/2019.
- Floudas, C. A. and Lin, X. (2004), 'Continuous-time versus discrete-time approaches for scheduling of chemical processes: A review', *Computers and Chemical Engineering* **28**(11), 2109–2129.
- Floudas, C. A. and Lin, X. (2005), 'Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications', *Annals of Operations Research* **139**(1), 131–162.
- Ford Jr, L. R. (1956), Network flow theory, Technical report, Rand Corp Santa Monica Ca.
- Garaix, T., Artigues, C., Feillet, D. and Josselin, D. (2010), 'Vehicle routing problems

- with alternative paths: An application to on-demand transportation', *European Journal of Operational Research* **204**(1), 62 – 75.
- Garey, M. R. and Johnson, D. S. (1990), *Computers and Intractability; A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., New York, NY, USA.
- Gendreau, M. and Potvin, J.-Y. (2005), 'Metaheuristics in combinatorial optimization', *Annals of Operations Research* **140**(1), 189–213.
- Glover, F. (1977), 'Heuristics for integer programming using surrogate constraints', *Decision sciences* **8**(1), 156–166.
- Glover, F. (1986), 'Future paths for integer programming and links to artificial intelligence', *Computers operations research* **13**(5), 533–549.
- Glover, F. (1989), 'Tabu search – part I', *ORSA Journal on computing* **1**(3), 190–206.
- Gopalan, R. and Talluri, K. T. (1998), 'The Aircraft Maintenance Routing Problem', *Operations Research* **46**(2), 260–271.
- Grall, A., Bérenguer, C. and Dieulle, L. (2002), 'A condition-based maintenance policy for stochastically deteriorating systems', *Reliab. Eng. Syst. Saf.* **76**(2), 167–180.
- Grall, A., Dieulle, L., Bérenguer, C. and Roussignol, M. (2002), 'Continuous-time predictive-maintenance scheduling for a deteriorating system', *IEEE Trans. Reliab.* **51**(2), 141–150.
- Grönkvist, M. (2005), *The tail assignment problem*, PhD thesis, Göteborg University.

- Gschwind, T., Irnich, S., Rothenbächer, A.-K. and Tilk, C. (2018), ‘Bidirectional labeling in column-generation algorithms for pickup-and-delivery problems’, *European Journal of Operational Research* **266**(2), 521–530.
- Gürkan, H., Gürel, S. and Aktürk, M. S. (2016), ‘An integrated approach for airline scheduling, aircraft fleet and routing with cruise speed control’, *Transp. Res. Part C Emerg. Technol.* **68**, 38–57.
- Gurobi Optimization version 8.1, I. (2019), ‘Gurobi Optimizer Reference Manual’. Accessed: 14/08/2019.
- Habibi, F., Barzinpour, F. and Sadjadi, S. (2018), ‘Resource-Constrained Project Scheduling Problem: Review of Past and Recent Developments’, *Journal of Project Management* **3**(2), 55–88.
- Hane, C. A., Barnhart, C., Johnson, E. L., Marsten, R. E., Nemhauser, G. L. and Sigismondi, G. (1993), ‘The fleet assignment problem: solving a large-scale integer program’, *Mathematical Programming* **70**(1-3), 211–232.
- Haouari, M., Shao, S. and Sherali, H. D. (2013), ‘A Lifted Compact Formulation for the Daily Aircraft Maintenance Routing Problem’, *Transportation Science* **47**(4), 508–525.
- Hart, P. E., Nilsson, N. J. and Raphael, B. (1968), ‘A formal basis for the heuristic determination of minimum cost paths’, *IEEE transactions on Systems Science and Cybernetics* **4**(2), 100–107.

- Harvey, W. D. and Ginsberg, M. L. (1995), Limited discrepancy search, *in* ‘IJCAI (1)’, pp. 607–615.
- Heng, A., Zhang, S., Tan, A. C. and Mathew, J. (2009), ‘Rotating machinery prognostics: State of the art, challenges and opportunities’, *Mech. Syst. Signal Process.* **23**(3), 724–739.
- Hicks, R., Madrid, R., Milligan, C., Pruneau, R., Kanaley, M., Dumas, Y., Lacroix, B., Desrosiers, J. and Soumis, F. (2005), ‘Bombardier Flexjet Significantly Improves Its Fractional Aircraft Ownership Operations’, *Interfaces* **35**(1), 49–60.
- Holland, J. (1975), ‘Adaptation in natural and artificial systems: an introductory analysis with application to biology’, *Control and artificial intelligence* .
- IATA (2019), [AVIATION DATA WHITE PAPER SERIES](#), Technical report, INTERNATIONAL AIR TRANSPORT ASSOCIATION (IATA).
- Irnich, S. (2008), ‘Resource extension functions: Properties, inversion, and generalization to segments’, *OR Spectrum* **30**(1), 113–148.
- Irnich, S. and Desaulniers, G. (2005), *Column Generation*, Springer US, chapter Shortest Path Problems with Resource Constraints, pp. 33–65.
- Irnich, S. and Villeneuve, D. (2006), ‘The Shortest-Path Problem with Resource Constraints and k -Cycle Elimination for $k \geq 3$ ’, *INFORMS Journal on Computing* **18**(3), 391–406.

- Jardine, A. K. S., Lin, D. and Banjevic, D. (2006), ‘A review on machinery diagnostics and prognostics implementing condition-based maintenance’.
- Jarrah, A. I., Yu, G., Krishnamurthy, N. and Rakshit, A. (1993), ‘A decision support framework for airline flight cancellations and delays’, *Transportation Science* **27**(3), 266–280.
- Jayabalan, V. and Chaudhuri, D. (1992), ‘Cost optimisation of maintenance scheduling for a system with assured reliability’, *IEEE Trans. Reliab.* **41**(1), 21–25.
- Jhwei, R., Aranki, N., Feldkamp, L. and Marko, K. (2008), ‘Ultra-compact neuroprocessor for automotive diagnostics and control’.
- Keller, R. E. and Poli, R. (2007), Cost-benefit investigation of a genetic-programming hyperheuristic, in ‘International Conference on Artificial Evolution (Evolution Artificielle)’, Springer, pp. 13–24.
- Kenan, N., Jebali, A. and Diabat, A. (2018), ‘The integrated aircraft routing problem with optional flights and delay considerations’, *Transportation Research Part E: Logistics and Transportation Review* **118**, 355–375.
- Kennedy, J. and Eberhart, R. (1995), Particle swarm optimization, in ‘Proceedings of ICNN’95-International Conference on Neural Networks’, Vol. 4, IEEE, pp. 1942–1948.
- Khaled, O., Minoux, M., Mousseau, V., Michel, S. and Ceugniet, X. (2018), ‘A Compact Optimization Model for the Tail Assignment Problem’, *European Journal of Operational Research* **264**(2), 548–557.

- Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P. (1983), ‘Optimization by simulated annealing’, *science* **220**(4598), 671–680.
- Kohl, N. and Karisch, S. E. (2004), ‘Airline crew rostering: Problem types, modeling, and optimization’, *Annals of Operations Research* **127**(1-4), 223–257.
- Kolisch, R. and Sprecher, A. (1997), ‘PSPLIB - A project scheduling problem library: OR Software - ORSEP Operations Research Software Exchange Program’, *European Journal of Operational Research* **96**(1), 205 – 216.
- Koné, O., Artigues, C., Lopez, P. and Mongeau, M. (2011), ‘Event-based MILP models for resource-constrained project scheduling problems’, *Computers & Operations Research* **38**, 3–13.
- Koné, O., Artigues, C., Lopez, P. and Mongeau, M. (2013), ‘Comparison of mixed integer linear programming models for the resource-constrained project scheduling problem with consumption and production of resources’, *Flexible Services and Manufacturing Journal* **25**(1-2), 25–47.
- Kopanos, G. M., Kyriakidis, T. S. and Georgiadis, M. C. (2014), ‘New continuous-time and discrete-time mathematical formulations for resource-constrained project scheduling problems’, *Computers & Chemical Engineering* **68**, 96 – 106.
- Kumar, R., Joshi, A. H., Banka, K. K. and Rockett, P. I. (2008), Evolution of hyperheuristics for the biobjective 0/1 knapsack problem by multiobjective genetic programming, in ‘Proceedings of the 10th annual conference on Genetic and evolutionary computation’, ACM, pp. 1227–1234.

- Lagerholm, M., Peterson, C. and Sderberg, B. (2000), ‘Airline crew scheduling using Potts mean field techniques’, *European Journal of Operational Research* **120**(1), 81–96.
- Lee, L. H., Lee, C. U. and Tan, Y. P. (2006), ‘A multi-objective genetic algorithm for robust flight scheduling using simulation’, *European Journal of Operational Research* **177**(3), 1948–1968.
- Letovsky, L. (1997), *Airline operations recovery: An optimization approach.*, PhD thesis, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA.
- Letovsky, L., Johnson, E. L. and Nemhauser, G. L. (2000), ‘Airline Crew Recovery’, *Transportation Science* **34**(4), 337–348.
- Levin, A. (1971), ‘Scheduling and Fleet Routing Models for Transportation Systems’, *Transportation Science* **5**(3), 232–255.
- Li, H., Wang, Y., Li, S. and Li, S. (2015), ‘A column generation based hyper-heuristic to the bus driver scheduling problem’, *Discrete Dynamics in Nature and Society* **2015**.
- Li, Z., Guo, J. and Zhou, R. (2016), Maintenance scheduling optimization based on reliability and prognostics information, *in* ‘Annual Reliability and Maintainability Symposium (RAMS)’, pp. 1–5.
- Liang, Z. and Chaovaitwongse, W. (2013), ‘A Network-Based Model for the In-

- tegrated Weekly Aircraft Maintenance Routing and Fleet Assignment Problem’, *Transportation Science* **47**, 493–507.
- Liang, Z., Xiao, F., Qian, X., Zhou, L., Jin, X., Lu, X. and Karichery, S. (2018), ‘A column generation-based heuristic for aircraft recovery problem with airport capacity constraints and maintenance flexibility’, *Transportation Research Part B: Methodological* **113**, 70–90.
- Liu, T.-K., Jeng, C.-R. and Chang, Y.-H. (2008), ‘Disruption management of an inequality-based multi-fleet airline schedule by a multi-objective genetic algorithm’, *Transportation Planning and Technology* **31**(6), 613–639.
- Løve, M., Sørensen, K., Larsen, J. and Clausen, J. (2001), ‘Using heuristics to solve the dedicated aircraft recovery problem’, *Optimization Online* .
- Lozano, L., Duque, D. and Medaglia, A. L. (2015), ‘An Exact Algorithm for the Elementary Shortest Path Problem with Resource Constraints’, *Transportation Science* **50**(1), 348–357.
- Maher, S. J. (2015), ‘A novel passenger recovery approach for the integrated airline recovery problem’, *Computers & Operations Research* **57**, 123–137.
- Maher, S. J. (2016), ‘Solving the Integrated Airline Recovery Problem Using Column-and-Row Generation’, *Transportation Science* **50**(1), 216–239.
- Maher, S. J., Desaulniers, G. and Soumis, F. (2014), ‘Recoverable robust single day aircraft maintenance routing problem’, *Computers & Operations Research* **51**, 130–145.

- Maher, S. J., Desaulniers, G. and Soumis, F. (2018), ‘The daily tail assignment problem under operational uncertainty using look-ahead maintenance constraints’, *European Journal of Operational Research* **264**(2), 534–547.
- Marinakis, Y., Migdalas, A. and Sifaleras, A. (2017), ‘A hybrid particle swarm optimization–variable neighborhood search algorithm for constrained shortest path problems’, *European Journal of Operational Research* **261**(3), 819–834.
- Marla, L., Vaaben, B. and Barnhart, C. (2016), ‘Integrated disruption management and flight planning to trade off delays and fuel burn’, *Transportation Science* **51**(1), 88–111.
- Marques, G., Nesello, V., Vanderbeck, F., Tahiri, I., Bulhões, T. and Sadykov, R. (2020), ‘`coluna.jl` – branch-and-price-and-cut in julia’.
- Marseguerra, M., Zio, E. and Podofillini, L. (2002), ‘Condition-based maintenance optimization by means of genetic algorithms and Monte Carlo simulation’, *Reliab. Eng. Syst. Saf.* **77**(2), 151–166.
- Martins, P. M. S. (2016), *Systematization and optimization of Fokker 100 s Maintenance Plan*, Technical report, Instituto Superior Técnico, Lisboa, Portugal.
- Mercier, A., Cordeau, J. F. and Soumis, F. (2005), ‘A computational study of Benders decomposition for the integrated aircraft routing and crew scheduling problem’, *Computers and Operations Research* **32**(6), 1451–1476.
- Mercier, A. and Soumis, F. (2005), ‘An integrated aircraft routing, crew scheduling and flight retiming model’, *Computers and Operations Research* **34**(8), 2251–2265.

- Minkowski, H. (1953), ‘Geometrie der Zahlen, Teubner, Leipzig, (1896)’, *Reprint: H. Minkowski, Geometrie der Zahlen, Chelsea*.
- Mirjalili, S. (2015), ‘Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm’, *Knowledge-based systems* **89**, 228–249.
- Mueller, E. and Chatterji, G. B. (2002), *Analysis of aircraft arrival and departure delay characteristics*, in ‘AIAA’s Aircraft Technology, Integration, and Operations (ATIO)’, pp. 1–14.
- Muter, I., Birbil, Ş. İ. and Bülbül, K. (2013), ‘Simultaneous column-and-row generation for large-scale linear programs with column-dependent-rows’, *Mathematical Programming* **142**(1-2), 47–82.
- Naber, A. (2017), ‘Resource-Constrained Project Scheduling with Flexible Resource Profiles in Continuous Time’, *Computers & Operations Research* **84**, 33–45.
- Nguyen, H. T. and Bhanu, B. (2012), *Zombie Survival Optimization: A swarm intelligence algorithm inspired by zombie foraging*, in ‘Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)’, IEEE, pp. 987–990.
- Nocedal, J. and Wright, S. (2006), *Numerical Optimization: Springer Series in Operations Research and Financial Engineering*, Springer.
- Ochoa, G., Walker, J., Hyde, M. and Curtois, T. (2012), *Adaptive evolutionary algorithms and extensions to the hyflex hyper-heuristic framework*, in ‘International Conference on Parallel Problem Solving from Nature’, Springer, pp. 418–427.

- Odili, J. B., Kahar, M. and Nizam, M. (2016), ‘Solving the traveling salesmans problem using the african buffalo optimization’, *Computational intelligence and neuroscience* **2016**.
- Orhan, I., Kapanoglu, M. and Karakoc, T. H. (2012), ‘Concurrent Aircraft Routing and Maintenance Scheduling Using Goal Programming’, *Journal of the Faculty of Engineering and Architecture of Gazi University* **27**(1), 11–26.
- Özcan, E., Misir, M., Ochoa, G. and Burke, E. K. (2010), ‘A Reinforcement Learning - Great-Deluge Hyper-Heuristic for Examination Timetabling’, *International Journal of Applied Metaheuristic Computing (IJAMC)* **1**(1), 39–59.
- Palpant, M., Boudia, M., Robelin, C., Gabteni, S. and Laburthe, F. (2009), ‘ROADEF 2009 challenge: Disruption management for commercial aviation’, *Sophia Antipolis, France: Operations Research Division* .
- Papadakos, N. (2009), ‘Integrated airline scheduling’, *Computers and Operations Research* **36**(1), 176–195.
- Papakostas, N., Papachatzakis, P., Xanthakis, V., Mourtzis, D. and Chryssolouris, G. (2010), ‘An approach to operational aircraft maintenance planning’, *Decis. Support Syst.* **48**(4), 604–612.
- Pecin, D., Pessoa, A., Poggi, M. and Uchoa, E. (2017), ‘Improved branch-cut-and-price for capacitated vehicle routing’, *Mathematical Programming Computation* **9**(1), 61–100.

- Peng, Z. and Chu, F. (2004), ‘Application of the wavelet transform in machine condition monitoring and fault diagnostics: a review with bibliography’, *Mechanical Systems and Signal Processing* **18**(2), 199–221.
- Pessoa, A., Sadykov, R., Uchoa, E. and Vanderbeck, F. (2019), A generic exact solver for vehicle routing and related problems, in ‘International Conference on Integer Programming and Combinatorial Optimization’, Springer, pp. 354–369.
- Petersen, J. D., Snelting, G., Clarke, J.-P., Johnson, E. L. and Shebalov, S. (2012), ‘An Optimization Approach to Airline Integrated Recovery’, *Transportation Science* **46**(4), 482–500.
- Polya, G. (1945), *How to solve it: A new aspect of mathematical method*, Princeton university press.
- Qantas (2016), ‘Qantas News Room: The A, C and D of Aircraft Maintenance.’. Accessed: 14/08/2019.
- Righini, G. and Salani, M. (2006), ‘Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints’, *Discrete Optimization* **3**(3), 255–273.
- Righini, G. and Salani, M. (2008), ‘New dynamic programming algorithms for the resource constrained elementary shortest path problem’, *Networks* **51**(3), 155–170.
- Rolls-Royce Holdings plc (2015), ‘Annual Report’. Accessed: 14/08/2019.

- Rolls-Royce Holdings plc (2016), ‘Civil Aerospace: Service Solutions’. Accessed: 14/08/2019.
- Rolls-Royce Holdings plc (2017), ‘Rolls-Royce Data Storytelling Challenge’. Accessed: 22/08/2019.
- Runka, A. (2009), Evolving an edge selection formula for ant colony optimization, *in* ‘Proceedings of the 11th Annual conference on Genetic and evolutionary computation’, ACM, pp. 1075–1082.
- Sadykov, R., Tahiri, I., Vanderbeck, F., Pessoa, A., Bulhões, T. and Marques, G. (2019), ‘VRPSolver – Branch-Cut-and-Price solver for vehicle routing and related problems.’.
- Sadykov, R., Vanderbeck, F., Pessoa, A., Tahiri, I. and Uchoa, E. (2019), ‘Primal Heuristics for Branch and Price: The Assets of Diving Methods’, *INFORMS Journal on Computing* **31**(2), 251–267.
- Safaei, N. and Jardine, A. K. (2018), ‘Aircraft routing with generalized maintenance constraints’, *Omega* **80**, 111 – 122.
- Salazar-González, J. J. (2014), ‘Approaches to solve the fleet-assignment, aircraft-routing, crew-pairing and crew-rostering problems of a regional carrier’, *Omega (United Kingdom)* **43**, 71–82.
- Sarac, A., Batta, R. and Rump, C. M. (2006), ‘A branch-and-price approach for operational aircraft maintenance routing’, *European Journal of Operational Research* **175**(3), 1850–1869.

- Sousa, J. P. and Wolsey, L. A. (1992), ‘A time indexed formulation of non-preemptive single machine scheduling problems’, *Mathematical Programming* **54**(1-3), 353–367.
- Sriram, C. and Haghani, A. (2003), ‘An optimization model for aircraft maintenance scheduling and re-assignment’, *Transp. Res. Part A Policy Pract.* **37**(1), 29–48.
- johnjforrest et al.
- johnjforrest, Vigerske, S., Gambini Santos, H., Ralphs, T., Hafer, L., Kristjansson, B., jpfasano, EdwinStraver, Lubin, M., rlougee, jpgoncal1, h-i-gassmann and Saltzman, M. (2020), ‘coin-or/Cbc: Version 2.10.5’.
- Thengvall, B. G., Bard, J. F. and Yu, G. (2000), ‘Balancing user preferences for aircraft schedule recovery during irregular operations’, *Iie Transactions* **32**(3), 181–193.
- Thengvall, B. G., Yu, G. and Bard, J. F. (2001), ‘Multiple fleet aircraft schedule recovery following hub closures’, *Transportation Research Part A: Policy and Practice* **35**(4), 289–308.
- Tilk, C., Rothenbächer, A. K., Gschwind, T. and Irnich, S. (2017), ‘Asymmetry matters: Dynamic half-way points in bidirectional labeling for solving shortest path problems with resource constraints faster’, *European Journal of Operational Research* **261**(2), 530–539.
- Torres Sanchez, D. (2020), ‘cspy: A Python package with a collection of algorithms for the (Resource) Constrained Shortest Path problem’, *Journal of Open Source Software* **5**(49), 1655.

- Torres Sanchez, D., Boyacı, B. and Zografos, K. G. (n.d.), ‘An Optimisation Framework for Airline Fleet Maintenance Scheduling with Tail Assignment Considerations’, *Transportation Research Part B: Methodological*.
- Urgaonkar, R., Kozat, U. C., Igarashi, K. and Neely, M. J. (2010), [Dynamic resource allocation and power management in virtualized data centers](#), in ‘Network Operations and Management Symposium, IEEE’, pp. 479–486.
- Van Steen, M. (2010), ‘An introduction to graph theory and complex networks’, **144**.
- Weyens, T. (2018), ‘[pylgrim – Elementary Shortest Path Problem with or without Resource Constraint](#)’. Accessed: 14/08/2019.
URL: <https://github.com/ToonWeyens/pylgrim>
- Yan, S. and Tseng, C.-H. (2002), ‘[A passenger demand model for airline flight scheduling and fleet routing](#)’, *Computers and Operations Research* **29**(11), 1559–1581.
- Yan, S., Yang, T. H. and Chen, H. H. (2004), ‘[Airline short-term maintenance manpower supply planning](#)’, *Transp. Res. Part A Policy Pract.* **38**(9-10), 615–642.
- Yen, J. Y. (1970), ‘[An algorithm for finding shortest routes from all source nodes to a given destination in general networks](#)’, *Quarterly of Applied Mathematics* **27**(4), 526–530.
- Zamorano, E. and Stolletz, R. (2017), ‘[Branch-and-price approaches for the Multi-period Technician Routing and Scheduling Problem](#)’, *European Journal of Operational Research* **257**(1), 55–68.

Zapata, J. C., Hodge, B. M. and Reklaitis, G. V. (2008), ‘The multimode resource constrained multiproject scheduling problem: Alternative formulations’, *AIChE Journal* **54**(8), 2101–2119.

Zoethout, J. (2015), The edge adjacency matrix of a graph, Bachelor thesis, Universiteit Utrecht.