공학석사학위논문

Approximate Dynamic Programming Approach for Airport Gate Assignment Problem

공항 게이트 할당 문제에 대한 근사 동적 계획법

2023 년 8 월

서울대학교 대학원

산업공학과

김 학 용

Approximate Dynamic Programming Approach for Airport Gate Assignment Problem

공항 게이트 할당 문제에 대한 근사 동적 계획법

지도교수 이 경 식

이 논문을 공학석사 학위논문으로 제출함

2023 년 6 월

서울대학교 대학원

산업공학과

김 학 용

김학용의 공학석사 학위논문을 인준함

2023 년 6 월

위 원 장	홍 성 필	(인)
부위원장	이 경 식	(인)
위 원	문 일 경	(인)

Abstract

Approximate Dynamic Programming Approach for Airport Gate Assignment Problem

Hakyong Kim
Department of Industrial Engineering
The Graduate School
Seoul National University

In real-world airport gate assignment problem (AGAP), the planning of flight-to-gate assignments involves more than a thousand of flights and is subject to frequent real-time adjustments. Thus, an efficient solution approach for AGAP is required for airport operation in practice. Here, we propose an approximate dynamic programming (ADP) approach for AGAP. In our ADP approach, value function is approximated by the interpolation of upper bound and lower bound of true value function with consideration of lookahead horizon. Heuristic algorithms and the linear programming relaxation values of integer programming (IP) models for AGAP are used for the upper bound and the lower bound, respectively. We first compare the bounds for several IP models and show that the pattern-based model provides the strongest bound, whose size is exponential to the input size. Next, we propose an efficient column generation method and ADP acceleration techniques to over the computational complexity arising when using the pattern-based model. The effectiveness and

practicality of our ADP approach were demonstrated by computational experiments.

Keywords: Airport Gate Assignment Problem, Column Generation, Approximate Dynamic Programming, Acceleration Techniques, Extended Formulation

Student Number: 2021-26136

Contents

Abstract **i**

Contents **v**

List of Tables **vi**

List of Figures **vii**

Chapter 1 Introduction **1**

1.1 Background 1

1.2 Literature Review 4

1.2.1 Airport Gate Assignment Problem 4

1.2.2 Related Problems 5

1.2.3 Approximate Dynamic Programming 7

1.3 Motivation and Contributions 9

1.4 Organization of the Thesis 10

Chapter 2 Dynamic Programming Formulation and Approximate Dynamic Programming Approach for AGAP **11**

2.1 Problem Definition 12

2.2 Dynamic Programming Formulation 13

2.3 Approximate Dynamic Programming Approach 16
2.4 IP Models for AGAP and Comparison of Bounds 20
2.4.1 Basic Model 20
2.4.2 Network Model 21
2.4.3 Pattern-based Model 22
2.4.4 Comparison of Bounds 24

Chapter 3 Approximate Dynamic Programming Approach using Pattern-based Model 28

3.1 Solution Approach for Action Evaluation Problem 29
3.1.1 Column Generation Method for Action Evaluation Problem . 29
3.1.2 Solution Approach for Subproblem 33
3.1.3 Multiple Column Generation Strategy 35
3.2 ADP Acceleration Techniques 37
3.2.1 Early Fixing 38
3.2.2 Reordering of Action Sequence 39
3.2.3 Early Cut-off 40
3.3 Implementation Details 42
3.3.1 Initialization 42
3.3.2 Column Inheritance 42
3.3.3 Updating Bounds & Termination Criterion 43

Chapter 4 Computational Experiments 45

4.1 Experiment Setting 46
4.2 Effects of Algorithmic Parameters of ADP$(\mathcal{P}, \eta, \tau)$ 48

4.2.1 Sensitivity Analysis on the Value of Interpolation Ratio . . . 48
4.2.2 Effects of the ADP Acceleration Techniques 49
4.2.3 Scalability Test with respect to Parameters τ, ϵ 52
4.3 Performance of ADP$(\mathcal{P}, \eta, \tau)$ 55
4.3.1 Test on Artificial Data 56
4.3.2 Test on Real-world Data 58

Chapter 5 Conclusion **60**

Bibliography **62**

국문초록 **69**

감사의 글 **70**

List of Tables

Table 2.1 Notation for the AGAP and the DP formulation 13

Table 2.2 Notation for the pattern-based model 23

Table 4.1 Instance generation parameter 46

Table 4.2 Performance comparison among various ADP parameters for $F500G40$ 52

Table 4.3 Performance comparison of various methods on artificial data 57

Table 4.4 Performance comparison of various methods on real-world data 59

List of Figures

Figure 1.1 Illustration of AGAP 1

Figure 3.1 Required Computation of ADP$(\mathcal{P}, \eta, \tau)$ at Stage t 37

Figure 4.1 Sensitivity analysis on the value of η 49

Figure 4.2 Computation time comparison among ADP acceleration schemes 51

Figure 4.3 Action and iteration comparison among ADP acceleration schemes 51

Figure 4.4 Performance comparison among various ADP parameters for $F100G8$ 53

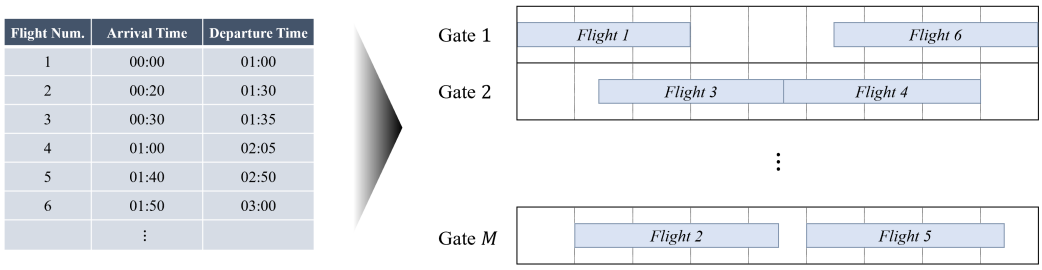
Figure 4.5 Performance comparison among various ADP parameters for $F250G20$ 53

Chapter 1

Introduction

1.1 Background

Establishing a good gate assignment plan is a crucial issue in airports as it has a significant impact on both the convenience of passengers and the operational efficiency of airlines/airports. Upon arrival at an airport, each aircraft requires allocation to an available gate to undergo necessary ground services such as embarking, cleaning, maintenance, and disembarking. However, gates are limited resources and coupled with other related operations in airports. Thus, efficient utilization of gates is necessary. Airport gate assignment problem (AGAP) is the planning of flight-to-gate assignments based on the scheduled arrival and departure times of flights. Because of the importance of AGAP, it has been studied for a long time in the optimization community. In recent years, several airport operation software products have



Flight Num.	Arrival Time	Departure Time
1	00:00	01:00
2	00:20	01:30
3	00:30	01:35
4	01:00	02:05
5	01:40	02:50
6	01:50	03:00
	⋮	

Figure 1.1: Illustration of AGAP

been developed to address AGAP. These products include DELMIA by Dassault Systèmes [1], as well as the airport operation system by Daifuku [2], among others.

There are two types of planning involved in AGAP [3, 4]: *in-advance planning* and *adaptive replanning*. In-advance planning is of establishing a gate assignment plan before operations commence based on the information of scheduled flights. In-advance planning depends on the policy of each airport. For instance, Incheon International Airport generates a gate assignment plan one day in advance using its own program based on the previously announced flight schedule (i.e., one day-ahead planning) [5]. Adaptive replanning, on the other hand, is of adjusting in-advance planning on the day of operation. In cases where applying in-advance planning is infeasible due to changes in scheduled flights, adjustments must be made to in-advance planning to account for these changes in the flight schedule [6].

Since there are various criteria for good planning in AGAP, diverse objectives and constraints have been considered [7, 8]. For passenger-oriented objectives, minimizing the total walking distance, discomfort, and waiting/transit time have been considered. For airlines/airport-oriented objectives, minimizing the number of ungated flights, total arrival delay, and operational cost have been considered. For constraints, two primary constraints are considered: assignment constraint and non-preemption constraint [9]. The assignment constraint states that each flight must be assigned to one gate while the non-preemption constraint states that only one flight can be processed at a gate at the same time. There are additional restrictions such as compatibility constraint according to aircraft types and airlines, as well as adjacency constraint, which prohibits the assignment of two heavy aircraft to adjacent gates simultaneously.

Among the various objectives of AGAP introduced above, arrival delay is one of the most important criteria that many stakeholders in airports pay careful attention to. As the interconnection between countries around the world continues to grow, arrival delays are very common nowadays and this phenomenon is expected to aggravate due to the growing congestion of air traffic. However, these delays are costly for both airlines/airports and their passengers [10, 11]. Delays incurred by airlines and airports result in substantial costs, primarily attributed to the crew, fuel, aircraft maintenance, and other operations. In addition, delays also have a significant impact on passengers, leading to reduced business productivity and missed opportunities for leisure activities due to extended air travel or waiting times.

In this thesis, we consider AGAP whose objective is of minimizing the total arrival delay. Based on the information on scheduled flights, the goal of the AGAP is to make an efficient gate assignment plan for both in-advance planning and adaptive replanning. However, AGAP is challenging both from a theoretical and a practical point of view [7]. In addition, there are two inherent difficulties in practice. Firstly, a practical problem size of an international airport is very large with more than a thousand flights per day. Secondly, arrival times and processing times are frequently changed in the real world and as a result, a predetermined gate assignment plan goes through frequent adaptive replannings. Thus, for a solution approach to be applicable in practice, it must be capable of solving large-scale AGAP in a reasonable amount of time. To this end, we propose an approximate dynamic programming (ADP) approach which can efficiently handle those two difficulties.

1.2 Literature Review

In this section, we discuss relevant research on arrival delay minimization AGAP and ADP approach, which is our solution method, in optimization problems. We include discussion on other problem domains, such as the parallel machine scheduling problem and the airport landing problem, since the AGAP can be interpreted in those areas. Findings and methodologies in other problem domains may be leveraged to inform the development of effective solution approaches to the AGAP.

1.2.1 Airport Gate Assignment Problem

In most research on AGAP, multiple objectives have been considered to accommodate the diverse criteria of AGAP. However, for the scope of this literature review, we focus on the objective of minimizing arrival delays. AGAP studies that consider the arrival delay of an aircraft were limited compared to other objective functions. Integer programming (IP) and meta-heuristics have been the main solution approaches. In [12], an evolutionary multi-objective optimization algorithm was proposed to minimize the total arrival delay and the number of un-gated flights in the sense of Pareto optimality. In their numerical experiments, the proposed method generated efficient solutions within hundreds of seconds for instances up to 5 gates and 100 flights, which is a fairly small size for an international airport. On the other hand, a column generation-based approach was proposed for an arrival delay minimization problem, where approximation algorithms and dynamic programming (DP) algorithms were utilized for solving subproblems [13]. The proposed solution approach could solve large-sized instances of the real world but requires a long computation time. To the best of our knowledge, [13] is the only study that considers arrival delay as a single

objective. For an in-depth review of AGAP studies, refer to [7, 8].

For IP models of AGAP, various formulations have been utilized. In a network model, a flight schedule on a single gate is represented by a path in a gate network. The network model has the advantage in that it can incorporate the problem-specific structure and additional constraints and thus, it has been widely used in AGAP research such as [14, 15]. On the other hand, in a pattern-based model, a flight schedule is defined by a pattern, satisfying restrictions within a gate. In general, the linear programming (LP) relaxation of a pattern-based model provides a bound that is at least as tight as the bound provided by the LP relaxation of a compact model. For this reason, a pattern-based model is preferred for exact algorithms [9, 13, 16].

1.2.2 Related Problems

Parallel Machine Scheduling Problem

AGAP can be viewed as a parallel machine scheduling problem (PMSP) with release dates. Specifically, flights and gates in AGAP correspond to jobs and machines in PMSP, respectively. The detailed relationship between AGAP and PMSP is explained in [17]. PMSP is a classic problem in operations research and computer science, and numerous methodologies have been developed to solve this problem. Among the various objective functions of PMSP, tardiness is equivalent to arrival delay in AGAP. In this literature review, we focus on tardiness minimizing PMSP with release dates.

For an exact method, the branch-and-price algorithm was proposed based on a set partitioning formulation [18]. The LP relaxation of a set partitioning problem is solved by the column generation method, where columns represent partial schedules

on single machines. Computational results showed that the LP relaxation value of the root node was very close to the optimal integer solution value. But the computation time of the column generation method increased exponentially with respect to the ratio of the number of jobs and the number of machines.

For heuristic methods, diverse dispatching rules have been suggested [17, 19]. Dispatching rule is a rule for determining which job should be assigned to which machine based on certain criteria. Earliest release date (ERD), earliest due date (EDD), shortest processing time (SPT), and longest processing time (LPT) are examples of the commonly used dispatching rules. In PMSP with tardiness minimization, apparent tardiness cost (ATC) rule, which is a composite dispatching rule, showed superior performance over other existing dispatching rules. Furthermore, several variations have been developed based on this ATC rule considering the specific characteristic of each problem [19]. For instance, when jobs have release dates, apparent tardiness cost with release date (ATCR) rule, a modification of ATC rule, was introduced in [20].

While dispatching rules can provide solutions quickly, they are often inadequate for complex PMSP in an application. In such cases, more sophisticated methods are required to achieve better performance. In [21], an iterated greedy meta-heuristic is developed for real-life production scheduling problems. In [22], on the other hand, two-stage stochastic programming was used to handle the uncertainties in job processing time and release time.

Aircraft Landing Problem

Aircraft landing problem (ALP), also known as airport runway scheduling problem, is the planning of the landing schedule of arriving aircraft to minimize total delays or other operational costs. Since runways are scarce resources that represent a bottleneck in many airports, extensive research has been conducted to efficiently utilize runways [23]. ALP can be interpreted as arrival delay minimization AGAP where runways in ALP correspond to gates in the AGAP.

Various solution approaches, such as DP [24, 25], branch-and-bound, and metaheuristics [26, 27, 28, 29], were proposed for ALP. The computational complexities of DP algorithms were suggested depending on the types of runways and aircraft classes [25]. In [24], ALP was formulated by DP where the state is defined as the number of assigned aircraft per class and runway occupation profile. In their DP formulation, the state space can be substantially reduced by dominance criteria, which leads to a dramatic reduction in computation time. DP formulation for ALP can be applied to AGAP with modification but DP approaches were not well utilized in the AGAP literature because of the problem-specific structure which hinders the reduction of search space in DP.

1.2.3 Approximate Dynamic Programming

ADP is a heuristic approach used to overcome the so-called *curse of dimensionality* in DP, which arises when the number of states or actions increases exponentially with respect to the problem size. ADP addresses this limitation by approximating the value function or policy function associated with the DP formulation. Policy is a rule (or function) that determines a decision given the available information in

a state [30]. For the past decade, ADP approaches have been widely used for solving complex and large-scale optimization problems because of their broad modeling capacity and algorithmic strategy [31]. Approximation strategies in the ADP framework can be categorized into four classes: cost function approximation, policy function approximation, lookahead approximation, value function approximation [30]. But these strategies can also be used in combination. References [31] and [32] cover general ADP approaches in detail and how they can be applied in practice. In this literature review, lookahead approximation and value function approximation are discussed, which are the key components in our methodology.

Lookahead approximation makes a decision of current stage by solving a problem over some horizon. A hybrid policy of lookahead approximation was proposed for a stochastic aircraft maintenance check scheduling problem in [33]. Dynamic lookahead policy, where lookahead horizons are parametrized by means of value function approximation, was proposed for a stochastic-dynamic inventory routing problem in [34]. A general framework for designing lookahead policies in transportation and logistic problems was suggested in [35].

In value function approximation, there are many ways for approximating the true value function, such as multilevel aggregation [36], basis function [37], and other statistical methods including the use of neural networks [38]. However, it can be approximated without any assumption on specific function structures. In [38] and [39], the interpolation of the upper bound and the lower bound of the true value function was used as the approximated value function for a deterministic multi-dimensional knapsack problem and a lot-sizing and scheduling problem, respectively.

1.3 Motivation and Contributions

Despite the growing necessity of research on arrival delay minimization AGAP, there have been limited studies focusing on this problem. Moreover, The previous solution approaches for the AGAP and its relevant problems, such as integer optimization, DP, and meth-heuristics, suffer from scalability issues, where computation time grows substantially with respect to problem size. As a result, current solution approaches for the AGAP are hardly applicable to real-world airports. At the same time, ADP approaches combined with optimization methodology have developed significantly over the past decade, showing good performances in many large-scale, complex optimization problems. However, as far as we know, there was no research on ADP approaches for AGAP. Therefore, we propose an optimization-based ADP approach that can efficiently solve large-scale AGAP. The main contributions of this thesis are as follows:

(a) We propose an ADP approach for AGAP. In the ADP approach, value function approximation combined with lookahead approximation is used.

(b) We compare the bounds of various IP models for AGAP. Through the comparison of bounds, we use a pattern-based model, which has the strongest bound, for computing a lower bound.

(c) We develop an efficient column generation method and ADP acceleration techniques to alleviate an excessive computational burden of the ADP algorithm.

1.4 Organization of the Thesis

The remainder of this thesis is organized as follows. In Chapter 2, we provide a DP formulation and propose an ADP approach for the AGAP. Also, we introduce three IP models, the basic model, network model and pattern-based model, and compare their bounds theoretically. In Chapter 3, a column generation method and ADP acceleration techniques are proposed for the ADP approach with the pattern-based model. The computational results for the ADP approach and the solution methods are discussed in Chapter 4. Chapter 5 concludes the thesis with a summary of the study and directions for future research.

Chapter 2

Dynamic Programming Formulation and Approximate Dynamic Programming Approach for AGAP

In this chapter, we explain a formal description of the AGAP and how this problem can be formulated as DP. However, the DP approach is impractical for the large-scale AGAP as the number of states and actions grows exponentially with respect to the problem size. To address this issue, we propose an ADP approach where the value function is approximated with consideration of a lookahead horizon. In the value function approximation, we utilize a dispatching rule and the LP relaxation value of an IP model. Three IP models for the AGAP are introduced and their bounds are theoretically compared.

2.1 Problem Definition

In this description, we use the word *flight* not only in its original meaning, but also to refer to the aircraft which corresponds to that flight. In the AGAP, we need to decide the assignments of N flights to M gates. Each flight $i \in F$ has an arrival time, h_i, and processing time, p_i. Here, the arrival time of a flight is the scheduled time of arrival at a gate and the processing time includes ground service time and buffer time. When flight i is assigned to a compatible gate $k \in G$, flight i needs parking at gate k at or after its arrival time. We refer to the time when flight i gets parked at its assigned gate as the *park time* of flight i. When a flight is parked at a gate, it occupies the gate for its processing time and no other flights can be processed at the gate during that time.

If every flight parks at its assigned gate precisely at its arrival time, no arrival delay is incurred. However, there are some situations when flights have to wait for available gates due to the congestion of a tight flight schedule. Then the arrival delay, the time interval between the arrival time and the park time, is incurred. The objective of the AGAP is to establish a flight-to-gate assignment plan that minimizes the total arrival delay. In this study, we do not consider adjacency restrictions since large aircraft do not occupy more than one gate in US airports and adjacency restrictions can be handled by compatibility constraints [8]. In addition, airport operations related to the parking of flights are not considered. This assumption can be justified because the flight-to-gate assignment is the most influential cause of arrival delay [13].

2.2 Dynamic Programming Formulation

Table 2.1: Notation for the AGAP and the DP formulation

Set	F	Set of flights, $i \in F = \{1, ..., N\}$
	G	Set of gates, $k \in G = \{1, ..., M\}$
	$\mathcal{S}_t$	Set of states at stage t
	$\mathcal{A}_t$	Set of actions at stage t
Function	$V_t(S_t)$	Value function of state S_t
	$C_t(S_t, a_t)$	Transition cost of state S_t and action a_t
Parameter	α_{ik}	1 if flight i is compatible with gate k, 0 otherwise
	p_i	Processing time of flight i
	h_i	Arrival time of flight i
	S_t	State at stage t; $S_t = ((c_t^k)_{k \in G}, d_t)$
	c_t^k	Completion time of gate k up to stage $t-1$
	d_t	Cumulative arrival delay up to stage $t-1$
Decision	a_t	Action at stage t; flight-to-gate assignment

The AGAP can be modeled as DP by assigning flights to gates one by one in order of arrival time sequentially. In the DP formulation, we assume that the planning horizon is discretized into time intervals of one minute. Notation for the DP formulation is presented in Table 2.1. The set of flights F is indexed in ascending order of arrival time, h_i. A stage, which represents the moment when a decision is made, is determined by the index of flights since exactly one flight is assigned to a gate for each stage. More specifically, at stage $t \in \{1, ..., T\}$, we determine one of the compatible gates that flight t can be assigned to. Thus, the last stage T is equal to the number of flights N.

A state S_t is defined by $(M+1)$-dimensional vector $((c_t^k)_{k \in G}, d_t)$ where c_t^k represents the completion time of gate k up to stage $t-1$ and d_t represents the cumulative arrival delays up to stage $t-1$. Since stage 0 is not defined, we define the initial state as $S_1 = ((c_1^k)_{k \in G}, d_1)$ where $c_1^k = 0$ for all $k \in G$ and $d_1 = 0$. An action $a_t = k$

represents the assignment of flight t to gate k. As there is a compatibility restriction between flights and gates, the set of possible actions, $\mathcal{A}_t = \{k \in G : \alpha_{tk} = 1\}$, can be different for each stage. When the state S_t is transitioned to the next state S_{t+1} by taking action a_t, a transition cost $C_t(S_t, a_t) = (c_t^{a_t} - h_t)^+$, also known as contribution function or reward, is incurred which is the amount of arrival delay of flight t. The transitioned state $S_{t+1} = ((c_{t+1}^k)_{k \in G}, d_{t+1})$ is updated as follows:

$$c_{t+1}^k = \begin{cases} \max\{c_t^k, h_t\} + p_t & \text{if} \quad k = a_t \\ c_t^k & \text{if} \quad k \neq a_t \end{cases}, \tag{2.1}$$

$$d_{t+1} = d_t + C_t(S_t, a_t). \tag{2.2}$$

Next, we define the value function $V_t(S_t)$, the value associated with the state S_t, by the minimum value of arrival delays of unassigned flights. Then, the value function can be written as

$$V_t(S_t) = \min_{\boldsymbol{a}_t \in \mathcal{X}_t} \left\{ \sum_{t'=t}^{N} C_{t'}(S_{t'}, a_{t'}) \middle| S_t \right\} \tag{2.3}$$

where $\mathcal{X}_t = \Pi_{t'=t}^{N} \mathcal{A}_{t'}$ and $\boldsymbol{a}_t = (a_t, ..., a_N)$. From the definition of $V_t(S_t)$, we can derive the following recursion

$$\begin{aligned} V_t(S_t) &= \min_{a_t \in \mathcal{A}_t} \left\{ C_t(S_t, a_t) + \min_{\boldsymbol{a}_{t+1} \in \mathcal{X}_{t+1}} \left\{ \sum_{t'=t+1}^{N} C_{t'}(S_{t'}, a_{t'}) \middle| S_{t+1} \right\} \right\} \\ &= \min_{a_t \in \mathcal{A}_t} \left\{ C_t(S_t, a_t) + V_{t+1}(S_{t+1}) \right\} \end{aligned} \tag{2.4}$$

for $t = 1, ..., N-1$ where S_{t+1} is the transitioned state from S_t by taking action a_t.

The optimal policy $\boldsymbol{a}^* = (a_1^*, ..., a_N^*)$ for AGAP is the flight-to-gate assignment solution that minimizes the total arrival delay

$$\boldsymbol{a}^* = \underset{\boldsymbol{a} \in \mathcal{X}}{\operatorname{argmin}} \left\{ \sum_{t=1}^{N} C_t(S_t, a_t) \middle| S_1 \right\} \tag{2.5}$$

where $\mathcal{X} = \Pi_{t=1}^{N} \mathcal{A}_t$ and $\boldsymbol{a} = (a_1, ..., a_N)$. The DP problem of equation (2.5) can be solved by the backward recursion algorithm using the recursion (2.4).

$$a_t^* = \underset{a_t \in \mathcal{A}_t}{\operatorname{argmin}} \left\{ C_t(S_t, a_t) + V_{t+1}(S_{t+1}) \right\} \tag{2.6}$$

However, due to the exponential number of states and actions with respect to the problem size, it is impractical to directly solve the DP problem.

2.3 Approximate Dynamic Programming Approach

Based on the DP formulation of the previous section, we propose an ADP approach for the AGAP. In the ADP approach, we use a value function approximation which is one of the most commonly used strategies in ADP framework. More specifically, we approximate state-action value function $v_t(S_t, a_t) = C_t(S_t, a_t) + V_t(S_{t+1})$, the value of action a_t taken at state S_t, as $\hat{v}_t(S_t, a_t)$. We refer to $\hat{v}_t(S_t, a_t)$ as *approximated value function*. Using this approximation, the action taken at stage t in equation (2.6) is replaced by

$$a_t^{ADP} = \underset{a_t \in \mathcal{A}_t}{\operatorname{argmin}}\ \hat{v}_t(S_t, a_t). \tag{2.7}$$

Contrary to the policy $\boldsymbol{a}^*$ in equation (2.5) which requires the evaluation of exponentially many future states, the policy $\boldsymbol{a}^{ADP} = (a_1^{ADP}, ..., a_N^{ADP})$ selects action greedily at each stage based solely on the approximated value function.

When designing the approximated value function $\hat{v}_t(S_t, a_t)$, it is important to capture the information of future states. However, it may be inefficient to consider all possible future states. The states for the near future are more important for current decision-making than the states for the distant future. For this reason, we introduce the concept of *lookahead horizon*, τ, to control the length of the period from the current stage when approximating $v_t(S_t, a_t)$. Specifically, up to τ upcoming flights from the current stage t are considered for evaluating $\hat{v}_t(S_t, a_t)$. The flight set $\{t, ..., \min\{t+\tau, N\}\}$ which is associated with the lookahead horizon period is called *lookahead flights*. When $\tau = 0$, actions are selected based only on the flight of the current stage, i.e. $a_t^{ADP} = \underset{a_t \in \mathcal{A}_t}{\operatorname{argmin}}\ C_t(S_t, a_t)$, because there is no consideration of

future flights. When $\tau = N$, $\hat{v}_t(S_t, a_t)$ considers all the unassigned flights from the current stage as with $v_t(S_t, a_t)$. Generally, as τ increases, the accuracy of $\hat{v}_t(S_t, a_t)$ improves, but the computational time for evaluating $\hat{v}_t(S_t, a_t)$ also increases. Thus, τ should be determined considering this trade-off between accuracy and computation time. Decision-making based on some horizon is called lookahead approximation in the ADP framework.

The approximated value function $\hat{v}_t(S_t, a_t)$ is defined by the interpolation of the upper bound $v_t^{UB}(S_t, a_t)$ and the lower bound $v_t^{LB}(S_t, a_t)$ of the true value of $v_t(S_t, a_t)$.

$$\hat{v}_t(S_t, a_t) = \eta \, v_t^{UB}(S_t, a_t) + (1 - \eta) \, v_t^{LB}(S_t, a_t), \quad \eta \in [0, 1] \tag{2.8}$$

The parameter η is the interpolation ratio of $v_t^{UB}(S_t, a_t)$ and $v_t^{LB}(S_t, a_t)$. If we can have the tighter bounds for $v_t(S_t, a_t)$, $\hat{v}_t(S_t, a_t)$ will be more accurate. But in general, obtaining tighter bounds for $v_t(S_t, a_t)$ necessitates more computation time. We use primal heuristics to obtain $v_t^{UB}(S_t, a_t)$ and the LP relaxation of IP models to obtain $v_t^{LB}(S_t, a_t)$. When evaluating $v_t^{UB}(S_t, a_t)$ and $v_t^{LB}(S_t, a_t)$, only the lookahead flights are considered. Pseudocode for the ADP algorithm with an instance $\mathcal{P}$ is as follows.

Algorithm 1 ADP($\mathcal{P}, \eta, \tau$)

1: $\boldsymbol{a}^{ADP} \leftarrow \emptyset$, $t \leftarrow 1$, $\boldsymbol{c} \leftarrow \boldsymbol{0}$, $d \leftarrow 0$; ▷ initialization
2: **while** $t \leq N$ **do**
3: $\quad a^{best} \leftarrow 0$, $\hat{v}^{best} \leftarrow \infty$; ▷ best action, best value function
4: $\quad$ **for** $a \in \mathcal{A}_t$ **do**
5: $\quad\quad v^{UB} \leftarrow heuristic(\mathcal{P}, \tau, t, \boldsymbol{c}, a)$;
6: $\quad\quad v^{LB} \leftarrow solveLP(\mathcal{P}, \tau, t, \boldsymbol{c}, a)$;
7: $\quad\quad \hat{v}^{cand} \leftarrow \eta\, v^{UB} + (1-\eta)\, v^{LB}$;
8: $\quad\quad$ **if** $\hat{v}^{cand} < \hat{v}^{best}$ **then**
9: $\quad\quad\quad a^{best} \leftarrow a$;
10: $\quad\quad\quad \hat{v}^{best} \leftarrow \hat{v}^{cand}$;
11: $\quad\quad$ **end if**
12: $\quad$ **end for**
13: $\quad \boldsymbol{a}^{ADP} \leftarrow \boldsymbol{a}^{ADP} \cup \{(t, a^{best})\}$;
14: $\quad \boldsymbol{c} \leftarrow updateCompletionTime(\mathcal{P}, t, \boldsymbol{c}, a^{best})$; ▷ Equation 2.1: completion time
15: $\quad d \leftarrow updateArrivalDelay(\mathcal{P}, t, \boldsymbol{c}, d, a^{best})$; ▷ Equation 2.2: arrival delay
16: $\quad t \leftarrow t + 1$;
17: **end while**
18: **return** $(d, \boldsymbol{a}^{ADP})$;

For primal heuristics of line 5 in ADP($\mathcal{P}, \eta, \tau$), we use dispatching rules in PMSP. The AGAP can be interpreted as tardiness minimization PMSP with both release date and due date. Release date and due date in PMSP correspond to arrival time (h_i) and arrival time plus processing time ($h_i + p_i$) in the AGAP, respectively. Among the various dispatching rules, we consider two dispatching rules, ERD and EDD since other dispatching rules such as SPT, LPT, ATCR are not suitable for the AGAP because of the problem-specific structure. The ERD rule first sorts the lookahead flights by the arrival time h_i's in ascending order. Then, the lookahead flights are assigned to compatible gates with the earliest completion time, $\underset{a_t \in \mathcal{A}_t}{\operatorname{argmin}}\, c_t^{a_t}$, sequentially. The EDD rule is similar to the ERD rule except that the lookahead flights are sorted by the arrival time plus processing time $h_i + p_i$'s in ascending order. In the pilot test for the ERD and EDD rules, the ERD rule showed better performance

compared to the EDD rule. Thus, we choose the ERD rule for the primal heuristic. For a given stage $\bar{t}$, completion time of gates $\bar{\boldsymbol{c}}$, action $\bar{a}$ in the ADP algorithm, pseudocode for the ERD rule algorithm is as follows.

Algorithm 2 ERD$(\mathcal{P}, \tau, \bar{t}, \bar{\boldsymbol{c}}, \bar{a})$

1: $\boldsymbol{a}^{ERD} \leftarrow \emptyset$, $\boldsymbol{c} \leftarrow \bar{\boldsymbol{c}}$, $d \leftarrow 0$, $t \leftarrow \bar{t}$; ▷ initialization
2: **while** $t \leq \min\{\bar{t} + \tau,\ N\}$ **do**
3: **if** $t = \bar{t}$ **then**
4: $a^* \leftarrow \bar{a}$;
5: **else**
6: $a^* \leftarrow \underset{a \in \mathcal{A}_t}{\operatorname{argmin}}\ c^a$; ▷ select the action with the earliest completion time
7: **end if**
8: $\boldsymbol{c} \leftarrow \mathit{updateCompletionTime}(\mathcal{P}, t, \boldsymbol{c}, a^*)$; ▷ Equation 2.1: completion time
9: $d \leftarrow \mathit{updateArrivalDelay}(\mathcal{P}, t, \boldsymbol{c}, d, a^*)$; ▷ Equation 2.2: arrival delay
10: $t \leftarrow t + 1$;
11: **end while**
12: **return** d ;

For IP models of line 6 in ADP$(\mathcal{P}, \eta, \tau)$, we consider three IP models for AGAP: basic model, network model, and pattern-based model. When these models are used for evaluating $v_t^{LB}(S_t, a_t)$, there are two distinctions from the IP models for the original AGAP. Firstly, partial flight-to-gate assignments, associated with the state S_t and the action a_t, are fixed. Secondly, only the lookahead flights are considered for the arrival delay. We refer to the problems for evaluating $v_t^{LB}(S_t, a_t)$ for all $S_t \in \mathcal{S}_t$, $a_t \in \mathcal{A}_t$, $t \in F$ as the *action evaluation problems* (EP). To better approximate the value function, we need to use an IP model with a strong LP relaxation value for EP. But there was no theoretical analysis of bounds among IP models for AGAP in the literature. In the next section, we introduce the three IP models for the AGAP and compare their bounds.

2.4 IP Models for AGAP and Comparison of Bounds

2.4.1 Basic Model

The basic model (B) includes explicit flight-to-gate assignment decision variables x_{ik}'s. x_{ik} equals to 1 if flight i is assigned to gate k, 0 otherwise. Decision variable t_i represents a park time of flight i to its assigned gate. The basic model has $O(NM)$ variables and $O(N^2M)$ constraints. The basic model is written as follows:

$$\text{(B):} \quad \min \quad \sum_{i \in F} (t_i - h_i) \tag{2.9}$$

$$\text{s.t.} \quad \sum_{k \in G} x_{ik} = 1, \quad \forall i \in F, \tag{2.10}$$

$$x_{ik} \leq \alpha_{ik}, \quad \forall i \in F, \forall k \in G, \tag{2.11}$$

$$t_i \geq h_i, \quad \forall i \in F, \tag{2.12}$$

$$t_i + p_i - t_j \leq U(2 - x_{ik} - x_{jk}), \quad \forall i < j, \forall i, j \in F, \forall k \in G, \tag{2.13}$$

$$x_{ik} \in \{0, 1\}, \quad \forall i \in F, \forall k \in G, \tag{2.14}$$

$$t_i \geq 0, \quad \forall i \in F \tag{2.15}$$

where U is a sufficiently large constant.

The objective function (2.9) represents the sum of arrival delays for all flights. Constraints (2.10) represent the assignment restriction. Constraints (2.11) represent the compatibility restriction. Constraints (2.12) indicate that flights can park at their assigned gates at or after their arrival time. Constraints (2.13) ensure that if flight i and j are assigned to a same gate, then subsequent flight j can park only after the processing time of preceding flight i.

2.4.2 Network Model

In the network model (N), there are M gate networks, where each network consists of nodes (representing flights) and arcs (representing precedence between flights) incorporating compatibility restrictions. Specifically, the k-th gate network consists of node set $NS_k = \{i \in F : \alpha_{ik} = 1\} \cup \{0, N+1\}$ and arc set $A_k = \{(i,j) : i \in NS_k,\ j \in NS_k \text{ s.t. } i < j\}$. Node 0 and node $N+1$ in NS_k are dummy nodes for starting and ending of flight schedule in gate k. A flow of arc $(i,j) \in A_k$ indicates that flight j is assigned immediately after flight i at gate k. The network model has been widely used in AGAP literature as many constraints of AGAP can be incorporated within the gate network [14, 15].

The network model uses decision variables y_{ij}^k's to represent flows in the gate networks. y_{ij}^k equals to 1 if flight j is assigned immediately after flight i at gate k, 0 otherwise. Since any path from node 0 to node $N+1$ corresponds to a flight schedule at a gate, the AGAP with the network model becomes a problem of finding the optimal paths in the gate networks. The network model has $O(N^2M)$ variables and $O(N^2 + NM)$ constraints. The network model is written as follows:

$$
\begin{aligned}
\text{(N):} \quad \min \quad & \sum_{i \in F} (t_i - h_i) && (2.16)\\
\text{s.t.} \quad & \sum_{i \in NS_k \setminus \{0\}} y_{0i}^k = 1, \quad \forall k \in G, && (2.17)\\
& \sum_{(i,j) \in A_k} y_{ij}^k = \sum_{(j,i) \in A_k} y_{ji}^k, \quad \forall i \in F,\ \forall k \in G, && (2.18)\\
& \sum_{i \in NS_k \setminus \{N+1\}} y_{i,N+1}^k = 1, \quad \forall k \in G, && (2.19)\\
& \sum_{k \in G} \sum_{(i,j) \in A_k} y_{ij}^k = 1. \quad \forall i \in F, && (2.20)\\
& t_i \geq h_i, \quad \forall i \in F, && (2.21)\\
& t_i + p_i - t_j \leq U \left(1 - \sum_{k \in G} y_{ij}^k \right), \quad \forall i < j,\ \forall i, j \in F, && (2.22)\\
& y_{ij}^k \in \{0, 1\}, \quad \forall (i,j) \in NS_k,\ \forall k \in G, && (2.23)\\
& t_i \geq 0, \quad \forall i \in F. && (2.24)
\end{aligned}
$$

The objective function (2.16) represents the sum of arrival delays for all flights. Constraints (2.17)-(2.19) represent the flow conservation in the gate networks. Constraints (2.20) represent the assignment restriction. Constraints (2.21)-(2.22) correspond to the constraints (2.12)-(2.13) in (B).

2.4.3 Pattern-based Model

In the pattern-based model (P), a flight schedule of a gate is represented by a pattern. Each pattern is defined by N-dimensional vector where each component indicates whether the corresponding flight is used or not. Given a pattern, we can easily recover the park time of flights and accordingly, the total arrival delay can be

calculated by the sum of the arrival delays of the patterns used.

Table 2.2: Notation for the pattern-based model

Set	P_k	Set of feasible patterns at gate k, $p \in P_k$
Parameter	δ_{ip}^k	1 if flight i is assigned on pattern $p \in P_k$ of gate k, 0 otherwise
	e_p^k	Arrival delay of pattern p
Decision	z_p^k	1 if pattern p is used at gate k, 0 otherwise

Constraints (2.11)-(2.13) in (B) can be incorporated within the pattern and thus, the pattern-based model can be formulated as a set partitioning problem. However, we used a set covering formulation for the pattern-based model as its LP relaxation is more numerically stable and it is trivial to construct an optimal solution of a set partitioning problem from a solution of a set covering problem [40]. The pattern-based model has $O(M2^N)$ variables and $O(N+M)$ constraints. Since there are an exponential number of patterns with respect to N, efficient solution approaches for solving the pattern-based model are needed [9, 13, 16]. The pattern-based model is written as follows:

$$\text{(P):} \quad \min \quad \sum_{k \in G} \sum_{p \in P_k} e_p^k z_p^k \tag{2.25}$$

$$\text{s.t.} \quad \sum_{k \in G} \sum_{p \in P_k} \delta_{ik}^k z_p^k \geq 1, \quad \forall i \in F, \tag{2.26}$$

$$\sum_{p \in P_k} z_p^k = 1, \quad \forall k \in G, \tag{2.27}$$

$$z_p^k \in \{0,1\}, \quad \forall p \in P_k,\ k \in G. \tag{2.28}$$

The objective function (2.25) represents the sum of arrival delays of the patterns used. Constrains (2.26) represent the assignment restriction. Constraints (2.27) ensure that only one pattern can be used for each gate.

2.4.4 Comparison of Bounds

Proposition 2.1. *Let $z(\mathrm{B})$ and $z(\mathrm{N})$ be the optimal objective values of the LP relaxations of* (B) *and* (N)*, respectively. Then,*

$$z(\mathrm{B}) \leq z(\mathrm{N}).$$

Proof. Let Q^B and Q^N be the sets of feasible solutions of the LP relaxation of (B) and (N), respectively. We will show that for any $(\hat{t}, \hat{y}) \in Q^N$, we can construct $(\hat{t}, \hat{x}) \in Q^B$. Let $y \in \mathbb{R}^{\frac{(N+2)(N+1)}{2}}$ be vector of flow from node 0 to node $N+1$ in the k-th gate network of (N). Then y can be decomposed by the sum of paths from node 0 to node $N+1$, i.e. $y = \sum_{p \in FL} f_p y^p$, where FL is the set of all possible flows, f_p is the amount of flow in path p and y^p is the characteristic vector of path p. For any path $p \in FL$ and a pair of flights (i, j) such that $i \in F,\ j \in F,\ i < j$, p belongs to one of the 4 sets:

$$FL_1 = \left\{ p \in FL : \sum_{i<q} y^p_{iq} = 0,\ \sum_{j<q} y^p_{jq} = 0 \right\},$$
$$FL_2 = \left\{ p \in FL : \sum_{i<q} y^p_{iq} > 0,\ \sum_{j<q} y^p_{jq} = 0 \right\},$$
$$FL_3 = \left\{ p \in FL : \sum_{i<q} y^p_{iq} = 0,\ \sum_{j<q} y^p_{jq} > 0 \right\},$$
$$FL_4 = \left\{ p \in FL : \sum_{i<q} y^p_{iq} > 0,\ \sum_{j<q} y^p_{jq} > 0 \right\},$$

where FL_1, FL_2, FL_3, FL_4 are mutually exclusive and collectively exhaustive for FL. For given the LP relaxation solution $(\hat{t}, \hat{y})$ for (N), define $\hat{x}_{ik} = \sum_{i<q} y^k_{iq}$, the

amount of flow passing the node i. For any $k \in G$, we have

$$\begin{aligned}
2 - \hat{x}_{ik} - \hat{x}_{jk} &= 1 + \left(\sum_{p \in FL} f_p - \sum_{p \in FL_2 \cup FL_4} f_p - \sum_{p \in FL_3 \cup FL_4} f_p \right) \\
&= 1 + \left(\sum_{p \in FL_1} f_p - \sum_{p \in FL_4} f_p \right) \\
&\geq 1 - \hat{y}_{ik}^k \\
&\geq 1 - \sum_{k \in G} \hat{y}_{ik}^k.
\end{aligned}$$

Since $U(2 - \hat{x}_{ik} - \hat{x}_{jk}) \geq U(1 - \sum_{k \in G} \hat{y}_{ik}^k) \geq \hat{t}_i + p_i - \hat{t}_j$, $(\hat{t}, \hat{x})$ satisfies the constraints (2.13) of (B). Therefore $(\hat{t}, \hat{x})$ is the solution of the LP relaxation of (B) and the LP relaxation value of (N) provides a bound that is at least as tight as the bound provided by (B). □

Proposition 2.2. *Let $z(\mathrm{N})$ and $z(\mathrm{P})$ be the optimal objective values of the LP relaxations of* (N) *and* (P)*, respectively. Then,*

$$z(\mathrm{N}) \leq z(\mathrm{P}).$$

Proof. Consider the set partitioning problem (P′) which is equivalent to (P) whose constraints (2.26) are substituted by equality constraints. Since (P) is the relaxation for (P′), it is sufficient to show that $z(\mathrm{N}) \leq z(\mathrm{P}')$. Let Q^N and $Q^{P'}$ be the sets of the feasible solutions of the LP relaxation of (N) and (P′), respectively. We will show that for any $\hat{z} \in Q^{P'}$, we can construct $(\hat{t}, \hat{y}) \in Q^N$.

For any $p \in P_k$, there exists a corresponding characteristic vector $y^{k,p} \in \mathbb{R}^{\frac{(N+2)(N+1)}{2}}$ in the kth gate network and it is defined by

$$y_{ij}^{k,p} = \left(\prod_{i<q<j} (1 - \delta_{qp}^k) \right) \delta_{ip}^k \delta_{jp}^k, \quad \forall i < j, \ \forall i, j \in \{0, ..., N+1\}.$$

where $\delta^k_{0p} = \delta^k_{N+1,p} = 1$ for all $p \in P_k$, $\forall k \in G$. For given the LP relaxation solution $\hat{z}$ for (P′), define $\hat{y} = (\hat{y}^1, ..., \hat{y}^M)$ by $\hat{y}^k_{ij} = \sum_{p \in P_k} y^{k,p}_{ij} \hat{z}^k_p$. From the definition of $y^{k,p}$, we have $\sum_{j<i} y^{k,p}_{ji} = 1$ for $i = 1, ..., N+1$ and $\sum_{j>i} y^{k,p}_{ij} = 1$ for $i = 0, ..., N$. Furthermore, we have

$$\sum_{j>i} \hat{y}^k_{ij} = \sum_{j>i} \sum_{p \in P_k} y^{k,p}_{ji} \hat{z}^k_p = \sum_{p \in P_k} \left(\sum_{j>i} y^{k,p}_{ji} \right) \hat{z}^k_p = \sum_{p \in P_k} \hat{z}^k_p = 1$$

and similarly, we can derive $\sum_{j<i} \hat{y}^k_{ji} = 1$. Thus $\hat{y}^k$ satisfies the flow conservation constraints (2.17) - (2.19).

For $p \in P_k$, the park time $t^{k,p}_i$ of flight i in pattern p is determined by

$$t^{k,p}_i = \max\Big\{ h_i, \max_{1 \le j < i} \{ t_j + p_j - U(1 - y^{k,p}_{ji}) \} \Big\}$$

and the corresponding cost coefficient is $c^k_p = \sum_{i \in F} (t^{k,p}_i - h_i)$.

Define $\hat{t}^{\mathrm{N}}_i = \sum_{k \in G} \sum_{p \in P_k} \delta^k_{ip} \hat{z}^k_p t^{k,p}_i$ for $i \in F$. Then we have

$$\text{(i)}: \quad \sum_{i \in F} (\hat{t}^{\mathrm{N}}_i - h_i) = \sum_{k \in G} \sum_{p \in P_k} \sum_{i \in F} \delta^k_{ip} \hat{z}^k_p (t^{k,p}_i - h_i) = \sum_{k \in G} \sum_{p \in P_k} c^k_p \hat{z}^k_p$$

and

$$\begin{aligned}
\text{(ii)}: \quad \hat{t}^{\mathrm{N}}_i &= \sum_{k \in G} \sum_{p \in P_k} \delta^k_{ip} \hat{z}^k_p \max\Big\{ h_i, \max_{1 \le j < i} \{ t_j + p_j - U(1 - y^{k,p}_{ji}) \} \Big\} \\
&= \max\Big\{ h_i, \sum_{k \in G} \sum_{p \in P_k} \delta^k_{ip} \hat{z}^k_p \max_{1 \le j < i} \{ t_j + p_j - U(1 - y^{k,p}_{ji}) \} \Big\} \\
&= \max\Big\{ h_i, \max_{1 \le j < i} \{ t^{\mathrm{N}}_j + p_j - U(1 - \sum_{k \in G} \sum_{p \in P_k} \delta^k_{ip} \hat{z}^k_p y^{k,p}_{ji}) \} \Big\} \\
&= \max\Big\{ h_i, \max_{1 \le j < i} \{ t^{\mathrm{N}}_j + p_j - U(1 - \sum_{k \in G} \hat{y}^k_{ji}) \} \Big\}.
\end{aligned}$$

From (ii), $(\hat{t}^{\mathrm{N}}, \hat{y})$ satisfies the constraints (2.21) - (2.22) in (N), and thus $(\hat{t}^{\mathrm{N}}, \hat{y}) \in Q^N$.

On the other hand, the objective function value of $(\hat{t}^{\mathrm{N}}, \hat{y})$ is equal to that of $\hat{z}$ in (P') by (i). Therefore $(\hat{t}^{\mathrm{N}}, \hat{y})$ is the solution of the LP relaxation of (N) and the LP relaxation value of (P') provides a bound that is at least as tight as the bound provided by (N). $\square$

Since Proposition (2.1) and Proposition (2.2) are about IP models for the original AGAP, we need the comparison of bounds among IP models for EP. For the convenience of notation, we denote the basic model, network model, and pattern-based model for EP by (EP-B), (EP-N), and (EP-P), respectively.

Corollary 2.3. *Let* $z(\text{EP-B})$*,* $z(\text{EP-N})$ *and* $z(\text{EP-P})$ *be the optimal objective values of the LP relaxations of* (EP-B)*,* (EP-N) *and* (EP-P) *for* EP*, respectively. Then,*

$$z(\text{EP-B}) \leq z(\text{EP-N}) \leq z(\text{EP-P}).$$

Corollary 2.3 can be proved similar to the proofs of Proposition (2.1) and Proposition (2.2). Based on Corollary 2.3, we choose (EP-P) for the evaluation of the lower bound $\hat{v}_t^{LB}(S_t, a_t)$ since it has the strongest LP relaxation value. Thus, we solve the LP relaxation problem of (EP-P) in $\mathrm{ADP}(\mathcal{P}, \eta, \tau)$ at line 6.

Chapter 3

Approximate Dynamic Programming Approach using Pattern-based Model

When the pattern-based model is used for the action evaluation problem, an excessive computation can be incurred in the ADP approach. In this chapter, we discuss solution approaches for the action evaluation problem and implementation details associated with the ADP algorithm. First, we explain a column generation method for the action evaluation problem. Then we develop several techniques that can accelerate the ADP algorithm. Finally, we elaborate on the implementation details of the ADP algorithm.

3.1 Solution Approach for Action Evaluation Problem

Column generation is an exact method commonly used to solve large-scale LP problems, especially with a large number of variables, and has been widely applied in various applications [41, 42]. The column generation method involves solving a restricted problem, which is a problem with a subset of columns, and generating new columns by solving subproblems. This procedure is repeated until no further profitable columns can be generated. A detailed explanation of the column generation method is introduced in [43]. In this section, we present a column generation method for the action evaluation problem, along with a solution approach for subproblems and a multiple column generation strategy.

3.1.1 Column Generation Method for Action Evaluation Problem

In the previous chapter, the pattern-based model (EP-P) was adopted for the action evaluation problem in the ADP approach. Thus the LP relaxation of (EP-P) has to be solved in line 6 of ADP$(\mathcal{P}, \eta, \tau)$. We denote this problem by $\mathrm{MP}_t(a_t)$ and refer to it as *master problem*. $\mathrm{MP}_t(a_t)$ is a LP problem for a given lookahead horizon τ, state S_t, and action a_t. We denote the lookahead flights by $F' = \{t, ..., \min\{t + \tau, N\}\}$. Although the notation in Table 2.2 was used for $\mathrm{MP}_t(a_t)$, its meaning is different. In $\mathrm{MP}_t(a_t)$, P_k represents the set of feasible patterns at gate k for flights in F'. e_p^k

represents the arrival delay of pattern $p \in P_k$. $\mathrm{MP}_t(a_t)$ can be written as follows:

$$\mathrm{MP}_t(a_t): \quad \min \quad \sum_{k \in G} \sum_{p \in P_k} e_p^k z_p^k \tag{3.1}$$

$$\text{s.t.} \quad \sum_{p \in P_k} \delta_{ip}^k z_p^k \geq 1, \quad i = t,\ k = a_t, \tag{3.2}$$

$$\sum_{k \in G} \sum_{p \in P_k} \delta_{ip}^k z_p^k \geq 1, \quad \forall i \in F' \setminus \{t\}, \tag{3.3}$$

$$\sum_{p \in P_k} z_p^k = 1, \quad \forall k \in G, \tag{3.4}$$

$$z_p^k \geq 0, \quad \forall p \in P_k,\ k \in G. \tag{3.5}$$

The objective function (3.1) represents the sum of arrival delays for the lookahead flights. Constraint (3.2) ensures that flight t is assigned to a gate corresponding to a given action a_t. Constraints (3.3)-(3.5) correspond to the constraints (2.26)-(2.28) in (P). The binary variable constraints (2.28) were replaced by constraints (3.5) due to the existence of constraints (3.4).

Directly solving $\mathrm{MP}_t(a_t)$ is impractical because it has exponentially many variables. Therefore, we use the column generation method where we start with a small subset of P_k and generate patterns gradually until an optimal solution is found. We denote a subset of P_k by $\widehat{P}_k$ and the master problem where P_k is replaced with $\widehat{P}_k$ by $\mathrm{RMP}_t(a_t)$. We refer to $\mathrm{RMP}_t(a_t)$ as *restricted master problem.* $\mathrm{RMP}_t(a_t)$ can be written as equations (3.6)-(3.10). After solving $\mathrm{RMP}_t(a_t)$, it is necessary to solve subproblems to determine whether profitable columns can be generated or not. Here, we define a column or pattern to be *profitable* if its corresponding variable, z_p^k, has a negative reduced cost. If profitable columns are not generated for all subproblems, the current solution of $\mathrm{RMP}_t(a_t)$ is the optimal solution of $\mathrm{MP}_t(a_t)$.

$$\text{RMP}_t(a_t): \quad \min \quad \sum_{k \in G} \sum_{p \in \widehat{P}_k} e_p^k z_p^k \tag{3.6}$$

$$\text{s.t.} \quad \sum_{p \in \widehat{P}_k} \delta_{ip}^k z_p^k \geq 1, \quad i = t,\ k = a_t, \tag{3.7}$$

$$\sum_{k \in G} \sum_{p \in \widehat{P}_k} \delta_{ip}^k z_p^k \geq 1, \quad \forall i \in F' \setminus \{t\}, \tag{3.8}$$

$$\sum_{p \in \widehat{P}_k} z_p^k = 1, \quad \forall k \in G, \tag{3.9}$$

$$z_p^k \geq 0, \quad \forall p \in \widehat{P}_k,\ k \in G. \tag{3.10}$$

Let π_i and μ_k be the dual optimal solutions of the constraints (3.7)-(3.8) and (3.9). Then, the problem of finding the most negative reduced cost is given by

$$\min\{e_p^k - \sum_{i \in F'} \delta_{ip}^k \pi_i - \mu_k : \forall p \in P_k,\ \forall k \in G\}. \tag{3.11}$$

This problem can be decomposed per gate and the term μ_k can be dropped since μ_k is a constant for each decomposed subproblem. We denote the decomposed subproblem for gate k by $\text{SP}_t^k(a_t)$. $\text{SP}_t^k(a_t)$ can be written as follows:

$$\text{SP}_t^k(a_t): \quad \min \quad \sum_{i \in F'} (u_i - h_i) - \sum_{i \in F'} x_{ik} \pi_i \tag{3.12}$$

$$\text{s.t.} \quad u_i \geq \max\{c_t^k, h_i\}, \quad \forall i \in F', \tag{3.13}$$

$$x_{ik} \leq \alpha_{ik}, \quad \forall i \in F', \tag{3.14}$$

$$u_i + p_i - u_j \leq U(2 - x_{ik} - x_{jk}), \quad \forall i < j,\ i, j \in F' \tag{3.15}$$

$$x_{ik} = \mathbb{1}_{\{k = a_t\}}, \quad i = t, \tag{3.16}$$

$$x_{ik} \in \{0, 1\}, \quad \forall i \in F', \tag{3.17}$$

$$u_i \geq 0, \quad \forall i \in F'. \tag{3.18}$$

The decision variables u_i's are used for the park time variables t_i's to distinguish them from the index of current stage t. $\mathrm{SP}_t^k(a_t)$ is a problem with respect to gate k with the following properties. First, only the lookahead flights are considered. Second, the park time u_i's of the lookahead flights F' cannot be earlier than the completion time c_t^k (constraint (3.13)). Third, the assignment of flight t is predetermined by the given action a_t (constraint (3.16)).

The solution $\boldsymbol{x}_k = (x_{ik}, ..., x_{lk})$ is the pattern generated by solving $\mathrm{SP}_t^k(a_t)$, where $l = \min\{t + \tau, N\}$. It is added to $\widehat{P}_k$ of $\mathrm{RMP}_t(a_t)$ when the optimal objective value of $\mathrm{SP}_t^k(a_t)$ is less than μ_k. The overall process of the column generation method for $\mathrm{MP}_t(a_t)$ is described in Algorithm 3. Algorithm 3 is executed in line 6 of $\mathrm{ADP}(\mathcal{P}, \eta, \tau)$.

Algorithm 3 Column generation for $\mathrm{MP}_t(a_t)$

```
generate initial columns for RMP_t(a_t) ;
repeat
    column_count ← 0 ;
    solve RMP_t(a_t) ;
    let z* and (π, μ) be the optimal value and dual solution of RMP_t(a_t) ;
    for k ∈ G do
        solve SP_t^k(a_t) ;
        let z_k* be the optimal value of SP_t^k(a_t) ;
        if z_k* − μ_k < 0 then
            add column to RMP_t(a_t) ;
            column_count ← column_count + 1 ;
        end if
    end for
until column_count = 0
return z* ;
```

3.1.2 Solution Approach for Subproblem

An efficient solution approach for subproblems is important in the column generation method, as a large number of subproblems have to be solved. $\mathrm{SP}_t^k(a_t)$ can be solved by a general mixed integer programming (MIP) solver. But due to the existence of a big number U in constraints (3.15), the performance of branch-and-bound algorithms in MIP solvers is very poor. Consequently, solving $\mathrm{SP}_t^k(a_t)$ by MIP solver requires a long computation time. Thus we use a DP algorithm of [13]. In [13], a DP algorithm was proposed for solving subproblems in a branch-and-price algorithm for the AGAP. However, since $\mathrm{SP}_t^k(a_t)$ is different from the subproblem in [13], we use the DP algorithm by modifying the input parameters of arrival time h_i and dual solution π_i. For the complete description of the solution approach for $\mathrm{MP}_t(a_t)$, we introduce the DP algorithm of [13] with modification of parameters.

Let $F'_k = \{1, 2, ..., n\}$ be the re-indexed set of $\{i \in F' \setminus \{t\} : \alpha_{ik} = 1 \text{ and } \pi_i > 0\}$. Flight t is excluded in F'_k for its assignment decision x_{ik} is already determined by the action a_t. Since flights with non-positive π_i cannot improve the objective function value of $\mathrm{SP}_t^k(a_t)$, it is sufficient to consider only the set F'_k for $\mathrm{SP}_t^k(a_t)$. Also, the parameter of arrival time, h_i, and dual solution, π_i, are re-indexed according to F'_k. Define $g_i(u)$ be the maximum total net benefit from optimally accepting flights in the set $\{i, i+1, ..., n\}$ at or after time u. A formal definition is given as follows:

$$g_i(u) = \max_{x_{jk} : j \geq i} \left\{ \sum_{j \geq i} \pi'_j x_{jk} - \sum_{j \geq i} (u_j - h'_j) \;\middle|\; u_j \geq u, \; \forall j \in \{i, i+1, ..., n\} \right\} \tag{3.19}$$

where h'_j, π'_j are adjusted arrival time, adjusted price defined by $h'_j = \max\{h_j, c_t^k\} + p_t \mathbb{1}_{\{k=a_t\}}$, $\pi'_j = \pi_j - (h'_j - h_j)$. We consider $u \in \{1, ..., c\}$ where c is the planning horizon for flight schedule in the subproblem. It suffices to choose c as $h'_n + \max_{1 \leq i \leq n} x_{ik}$.

If flight j is not assigned to gate k, i.e. $x_{jk} = 0$, then $u_j = h'_j$. Thus flight j does not contribute to the value of $g_i(u)$. On the other hand, if flight j is assigned to gate k, i.e. $x_{jk} = 1$, then $\pi'_j x_{jk} - (u_j - h'_j) = (\pi_j - (h'_j - h_j))x_{jk} - (u_j - h'_j) = \pi_j x_{jk} - (u_j - h_j)$. In this case, flight j contribute $\pi_j - (u_j - h_j)$ amount to the value of $g_i(u)$. Therefore, the optimal solution value for $\mathrm{SP}_t^k(a_t)$ is equal to $-g_1(h'_1) + (c_t^{a_t} - h_t)^+$ where $(c_t^{a_t} - h_t)^+$ is the arrival delay of flight t in the original flight set, F.

From the definition of $g_i(u)$, we can derive the following recursive formula:

$$g_i(u) = \begin{cases} 0 & \text{if} \quad i \geq n+1 \\ g_i(h'_i) & \text{if} \quad u < h'_i \\ g_{i+1}(u) & \text{if} \quad u > h'_i + \pi'_i \\ \max\{(h'_i + \pi'_i - u) + g_{i+1}(u + p_i), g_{i+1}(u)\} & \text{if} \quad h'_i \leq u \leq h'_i + \pi'_i \end{cases} \tag{3.20}$$

Based on the recursive formula, $g_1(h'_1)$ can be computed by the backward recursion algorithm of Algorithm 4. Algorithm 4 is executed in line 7 of Algorithm 3.

Algorithm 4 DP algorithm for $\mathrm{SP}_t^k(a_t)$

```
let g and S be a n+1 by c array
for u = c to h'_1 do
    g(n+1, u) ← 0 ;
    S(n+1, u) ← ∅ ;
end for
for i = n to 1 do
    for u = c to h'_1 do
        if u < h'_i then
            g(i, u) ← g(i, h'_i) ;
            S(i, u) ← S(i, h'_i) ;
        else if u > h'_i + π'_i then
            g(i, u) ← g(i+1, u), S(i, u) ← S(i+1, u) ;
            S(i, u) ← S(i+1, u) ;
            if h'_i + π'_i − u + g(i+1, u+p_i) > g(i+1, u) then
                g(i, u) ← h'_i + π'_i − u + g(i+1, u+p_i) ;
                S(i, u) ← S(i+1, u) ;
            else
                g(i, u) ← g(i+1, u) ;
                S(i, u) ← S(i+1, u) ;
            end if
        end if
    end for
end for
return (g, S) ;
```

3.1.3 Multiple Column Generation Strategy

In Algorithm 3, during each iteration, at most one pattern can be generated per gate. However, it may be inefficient in the column generation method where many patterns are needed to solve $\mathrm{MP}_t(a_t)$ to optimality. In this regard, a submodular maximization algorithm [44] was used to generate additional patterns in [13]. But this method can result in the inclusion of patterns with non-negative reduced costs, only to increase the size of $\mathrm{RMP}_t(a_t)$. Thus, we devise an alternative method, Algorithm 5, to generate multiple patterns per gate efficiently.

Algorithm 5 Multiple column generation algorithm

1: let (g, S) be the returned arrays of Algorithm 4 ;
2: U $\leftarrow \emptyset$;
3: **if** $n \geq 2$ and $\delta \geq 2$ **then**
4: **for** $i = 2, ..., \delta$ **do**
5: **if** $-g(i, h'_1) + (c_t^{a_t} - h_t)^+ - \mu_k < 0$ **then**
6: **if** $checkRedundancy(U, S(i, h'_1))$ **then**
7: $U \leftarrow \{S(i, h'_1)\}$;
8: **end if**
9: **end if**
10: **end for**
11: **end if**
12: add columns in U to $\mathrm{RMP}_t(a_t)$;

This method is, basically, based on the returned arrays (g, S) of Algorithm 4. For $i \in F'_k$ with $-g(i, h'_1) + (c_t^{a_t} - h_t)^+ - \mu_k < 0$, pattern $S(i, h'_1)$ is profitable (line 5). Adding these patterns to $\mathrm{RMP}_t(a_t)$ will speed up the column generation method. However, generating all profitable patterns can lead to an excessively large size of $\mathrm{RMP}_t(a_t)$. Hence, we exclude any patterns that are dominated by previously generated patterns (line 6). In addition, we set the maximum number of pattern generation by parameter δ (line 3). It suffices to consider $\delta \leq M$ and thus, we set $\delta = M$. Algorithm 5 is executed in line 10 of Algorithm 3.

3.2 ADP Acceleration Techniques

In the proposed ADP approach, a tremendous computational burden can be incurred for the following two reasons. Firstly, the number of action evaluations, specifically the number of $\mathrm{MP}_t(a_t)$'s needs to be solved, is proportional to N and M. Secondly, in the column generation method of $\mathrm{MP}_t(a_t)$, a number of iterations between $\mathrm{RMP}_t(a_t)$ and $\mathrm{SP}_t^k(a_t)$'s are required. Consequently, naively applying the ADP approach would take excessive computation time. To address this computational complexity, we develop several techniques which can accelerate the ADP algorithm based on the relationship between states.

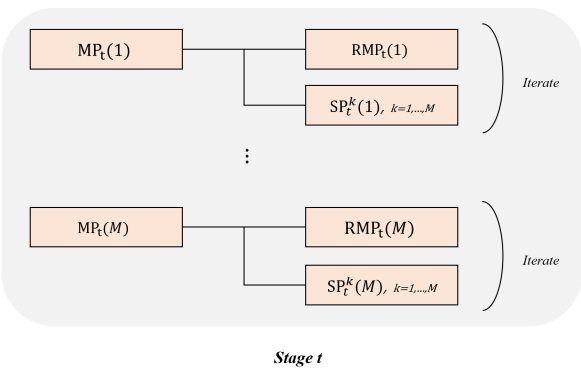


Figure 3.1: Required Computation of $\mathrm{ADP}(\mathcal{P}, \eta, \tau)$ at Stage t

There are three techniques for the ADP acceleration: Early Fixing (`EF`), Reordering of Action Sequence (`RAS`), and Early Cut-off (`EC`). `RAS` and `EF` are aimed at reducing both the number of action evaluations and iterations. `EC`, on the other hand, is aimed at reducing only the number of iterations. For the rest of this thesis, given IP or LP model (M), the optimal objective value of either the LP relaxation problem of (M) (in the case of an IP model) or the LP problem of (M) (in the case of an LP model) will be denoted as z(M).

3.2.1 Early Fixing

In the ADP approach, evaluating all candidate actions at each stage can be computationally expensive. To reduce the computation time, it is better to reduce the number of action evaluations if possible. `EF` is used to select the best action of the current stage without evaluating all candidate actions. More specifically, if the current action a_t satisfies a certain criterion, then we exit the for loop of lines 4-12 in $\mathrm{ADP}(\mathcal{P}, \eta, \tau)$.

`EF` uses the criterion that an action a_t is selected as the best action of the current stage if it satisfies the following inequality

$$d_t + \hat{v}_t(S_t, a_t) \leq (1 + \epsilon)\ (d_{t-1} + z(\mathrm{MP}_{t-1}(a_{t-1}^{ADP}))). \tag{3.21}$$

Here, $d_t + \hat{v}_t(S_t, a_t)$ represents the estimation of the sum of arrival delays from flight 1 to flight $\min\{t + \tau, N\}$. $d_{t-1} + z(\mathrm{MP}_{t-1}(a_{t-1}^{ADP}))$ represents the estimation of the sum of arrival delays from flight 1 to flight $\min\{t + \tau - 1, N\}$ and thus it becomes the lower bound for the value of $d_t + \hat{v}_t(S_t, a_t')$ for all $a_t' \in \mathcal{A}_t$. When `EF` is used in the ADP approach, the if statement for checking the inequality (3.21) is added between lines 11 and 12 in $\mathrm{ADP}(\mathcal{P}, \eta, \tau)$.

The non-negative parameter ϵ determines the threshold for `EF`. A smaller ϵ value allows for more possible actions to be evaluated, resulting in better solution quality but longer computation time. This is because it expands the search space, which can lead to better actions being discovered. Thus, there is a trade-off between solution quality and computation time in setting ϵ. When ϵ is set to 0, `EF` is performed only when the condition $d_t + \hat{v}_t(S_t, a_t) = d_{t-1} + z(\mathrm{MP}_{t-1}(a_{t-1}^{ADP}))$ is satisfied. In contrast,

when ϵ is set to ∞, the first action of the for loop is selected as the best action for the current stage, using an abuse of notation.

3.2.2 Reordering of Action Sequence

When `EF` is used in the ADP approach, it is advantageous to evaluate promising actions early in the stage. Promising actions are more likely to satisfy equation (3.21) which leads to fewer evaluations of candidate actions. For this reason, it is preferred to prioritize more promising actions. This is where `RAS` comes in: it specifies the order in which actions are evaluated at the current stage. That is, we sort the set of actions $\mathcal{A}_t$ in line 4 of $\text{ADP}(\mathcal{P}, \eta, \tau)$. There can be various criteria for sorting actions, but we present one naive criterion and two alternative criteria.

Naive criterion (`NAI`) sorts $a_t \in \mathcal{A}_t$ by the index of the gates. For example, for given $\mathcal{A}_t = \{1, 2, 3, 5\}$, the action sequence becomes $(a_{t_1}, a_{t_2}, a_{t_3}, a_{t_4}) = (1, 2, 3, 5)$. This criterion does not consider any information of the current state. The first alternative criterion is Earliest Available Gate (`EAG`). This criterion sorts $a_t \in \mathcal{A}_t$ by their completion time $c_t^{a_t}$ in ascending order. In other words, `EAG` criterion gives priority to actions that can start the processing of a new flight earlier than others. The second alternative criterion is Largest Assignment Solution (`LAS`). In this case, $a_t \in \mathcal{A}_t$ is sorted by the solution value $\hat{x}_{tk} = \sum_{p \in P_k} \delta_{tp}^k \hat{z}_p^k$ of $\text{MP}_{t-1}(a_{t-1}^{ADP})$ in descending order. That is, `LAS` criterion gives priority to actions that have the largest solution value $\hat{x}_{tk}$ at the previous stage.

Overall, by using these sorting criteria together with `EF`, we can significantly reduce the number of action evaluations required in the ADP algorithm.

3.2.3 Early Cut-off

In the column generation method for $\mathrm{MP}_t(a_t)$, the upper and lower bound for the optimal solution value can be updated during iterations. At intermediate iterations of $\mathrm{MP}_t(a_t)$, the upper and lower bound for $z(\mathrm{MP}_t(a_t))$ are given as

$$\text{Upper bound:} \quad z(\mathrm{RMP}_t(a_t)) \tag{3.22}$$

$$\text{Lower bound:} \quad z(\mathrm{RMP}_t(a_t)) + \sum_{k \in G} (z(\mathrm{SP}_t^k(a_t)) - \mu_k) \tag{3.23}$$

respectively [43]. These bounds can be used to determine if the current action a_t is the best action a_t^{ADP} of the current stage t before reaching the optimal solution. Using the lower bound, we can predetermine that the current action is not the best action. Using the upper bound, on the other hand, we can predetermine that the current action is the best action. In either case, the need to solve $\mathrm{MP}_t(a_t)$ exactly can be avoided and the evaluation of the current action can be stopped mid-process. This acceleration technique is referred to as `EC`. We call the first case of `EC` by *cut-off by lower bound* and the second case of `EC` by *cut-off by upper bound*.

Cut-off by Lower Bound

During the iteration of the column generation method of $\mathrm{MP}_t(a_t)$, if the lower bound of $\hat{v}_t(S_t, a_t)$ of the current action a_t is found to be greater than or equal to $\hat{v}_t(S_t, a_t')$ of a previously evaluated action $a_t' \in \mathcal{A}_t$, then we can predetermine that the current action is not the best action. Formally stating this condition, it is written as

$$\eta\, v_t^{UB}(S_t, a_t) + (1-\eta)\left(z(\mathrm{RMP}_t(a_t)) + \sum_{k \in G} (z(\mathrm{SP}_t^k(a_t)) - \mu_k)\right) \geq \hat{v}_t^{best} \tag{3.24}$$

where $\hat{v}_t^{best}$ is the smallest value of $\hat{v}_t(S_t, a'_t)$ of a previously evaluated action $a'_t \in \mathcal{A}_t$, which corresponds to $\hat{v}^{best}$ in ADP$(\mathcal{P}, \eta, \tau)$. The left-hand side of equation (3.24) represents the lower bound of $\hat{v}_t(S_t, a_t)$ since $z(\mathrm{RMP}_t(a_t)) + \sum_{k \in G}(z(\mathrm{SP}_t^k(a_t)) - \mu_k)$ is the lower bound for $v_t^{LB}(S_t, a_t)$. Satisfying equation (3.24) implies that $\hat{v}_t(S_t, a_t) \geq \hat{v}_t(S_t, a'_t)$ for some action $a'_t \in \mathcal{A}_t$ and thus we can predetermined that the current action a_t cannot be chosen as the best action a_t^{ADP}.

Cut-off by Upper Bound

During the iteration of the column generation method of $\mathrm{MP}_t(a_t)$, we can predetermine that the current action a_t is the best action if the following equation is satisfied:

$$d_t + \eta\, v_t^{UB}(S_t, a_t) + (1 - \eta)\, z(\mathrm{RMP}_t(a_t)) = d_{t-1} + z(\mathrm{MP}_{t-1}(a_{t-1}^{ADP})). \tag{3.25}$$

The left-hand side of equation (3.25) represents the upper bound of $d_t + \hat{v}_t(S_t, a_t)$ since $z(\mathrm{RMP}_t(a_t))$ is the upper bound of $v_t^{LB}(S_t, a_t)$. The right-hand side represents the lower bound for $d_t + \hat{v}_t(S_t, a'_t)$ for all $a'_t \in \mathcal{A}_t$, which is explained in section 3.2.1. Satisfying equation (3.25) implies that the left-hand side of equation (3.25) of current action a_t attains the lowest value of the current stage and thus we can predetermine that the current action can be chosen as the best action.

The if statements for checking the inequality (3.24) and equality (3.25) in the column generation algorithm in line 6 of ADP$(\mathcal{P}, \eta, \tau)$.

3.3 Implementation Details

In this section, we discuss technical issues regarding the implementation of the proposed ADP approach, which have a significant impact on the computation time of the overall algorithm. we explain initialization and column inheritance of the master problem $\mathrm{MP}_t(a_t)$. Then, we discuss how to update the upper and lower bounds of the solution value return by $\mathrm{ADP}(\mathcal{P}, \eta, \tau)$ at each stage. These bounds can be used for termination criterion of $\mathrm{ADP}(\mathcal{P}, \eta, \tau)$.

3.3.1 Initialization

When solving $\mathrm{MP}_t(a_t)$, specifically in line 6 of $\mathrm{ADP}(\mathcal{P}, \eta, \tau)$, initial feasible columns are needed for the restricted master problem to start the iteration of the column generation method. For initial columns, maximal columns considering compatibility with a sufficiently large cost coefficient are used. Also, columns generated by $\mathrm{ERD}(\mathcal{P}, \tau, t, \boldsymbol{c}, a)$, where columns can be constructed by tracking the flight-to-gate assignments, are additionally generated to speed up the convergence of column generation of $\mathrm{MP}_t(a_t)$.

3.3.2 Column Inheritance

Starting with high-quality columns can significantly reduce the number of iterations required to solve $\mathrm{MP}_t(a_t)$. Between $\mathrm{MP}_{t-1}(a_{t-1}^{ADP})$ and $\mathrm{MP}_t(a_t)$, the set of lookahead flights differ at most one element. Hence, $\mathrm{MP}_{t-1}(a_{t-1}^{ADP})$ and $\mathrm{MP}_t(a_t)$ share almost the same structure and it is expected that the set of generated columns for the two problems will be very similar. Therefore, we use the set of columns obtained from solving $\mathrm{MP}_{t-1}(a_{t-1}^{ADP})$ as the initial columns for $\mathrm{MP}_t(a_t)$ for all $a_t \in \mathcal{A}_t$ together with

the columns generated in subsection 3.3.1. We call this method *column inheritance*. Columns, i.e. patterns, in P_k of $\mathrm{MP}_{t-1}(a_{t-1}^{ADP})$ are inherited to columns in P_k of $\mathrm{MP}_t(a_t)$ for all $k \in G$, respectively. When $k = a_t$, only the columns satisfying the condition that flight t is assigned to gate a_t were inherited. It can be implemented by storing the set of patterns for all gates associated with $\mathrm{MP}_t(a^{best})$ in $\mathrm{ADP}(\mathcal{P}, \eta, \tau)$. To make column inheritance possible from stage 1, the LP relaxation of the pattern-based model with respect to the flight set $\{1, ..., \tau\}$ is solved before the while loop of line 2 in $\mathrm{ADP}(\mathcal{P}, \eta, \tau)$. At stage 1, columns obtained from solving this problem were inherited to $\mathrm{MP}_1(a_1)$ for all $a_1 \in \mathcal{A}_1$.

3.3.3 Updating Bounds & Termination Criterion

An *ADP solution* is a policy $\boldsymbol{a} = (a_1, ..., a_N)$ obtained by the proposed ADP approach. If the ADP algorithm is terminated within the time limit, then the ADP solution is given as $\boldsymbol{a}^{ADP} = (a_1^{ADP}, ..., a_N^{ADP})$. Otherwise, the best incumbent solution found so far is considered as the ADP solution. *ADP value* is defined as the arrival delay associated with the ADP solution. How to construct an incumbent solution and update the lower bound of the partial solution value at each stage is as follows.

At the beginning of each stage, we construct an incumbent solution and update the upper bound UB_t^{ADP} and lower bound LB_t^{ADP} of ADP value. The incumbent solution is given as $(a_1^{ADP}, ..., a_{t-1}^{ADP},\ a_t, ..., a_N)$ where $(a_1^{ADP}, ..., a_{t-1}^{ADP})$ is a partial solution constructed so far by the ADP algorithm and $(a_t, ..., a_N)$ is a partial solution for undetermined stages constructed by the ERD rule with the unassigned flights $\{t, ..., N\}$ not with lookahead flights. UB_t^{ADP} and LB_t^{ADP} are

different from $v_t^{UB}(S_t, a_t)$ and $v_t^{LB}(S_t, a_t)$ which are for a value function $v_t(S_t, a_t)$. UB_t^{ADP} is updated by $UB_t^{ADP} = \min\{UB_{t-1}^{ADP},\ W\}$, where W is the arrival delay of the incumbent solution $(a_1^{ADP}, ..., a_{t-1}^{ADP},\ a_t, ..., a_N)$. LB_t^{ADP} is updated by $LB_t^{ADP} = d_{t-1} + v_{t-1}^{LB}(S_{t-1}, a_{t-1}^{ADP})$. If UB_t^{ADP} and LB_t^{ADP} reach the same value, we can terminate the ADP algorithm before the last stage.

Chapter 4

Computational Experiments

In this chapter, we present the results of the computational experiments and discuss the effectiveness and efficiency of the proposed ADP approach. Experiments are mainly divided into two parts: effects of algorithmic parameters of $\mathrm{ADP}(\mathcal{P}, \eta, \tau)$ and performance of $\mathrm{ADP}(\mathcal{P}, \eta, \tau)$. In the first part of experiment, we investigate the behavior of the ADP approach with respect to the various control parameters. In the second part of experiment, we compare the performance of the ADP approach to other methods.

4.1 Experiment Setting

We conducted experiments on both artificial data and real-world data. To generate artificial data while controlling the amount of congestion of a gate, we introduced several parameters in Table 4.1 and followed the procedure provided by [9] similarly.

Table 4.1: Instance generation parameter

N	Number of flights
M	Number of gates, $\lfloor \frac{N\bar{p}}{T^H u} \rfloor$
u	Congestion of gate, $u \in (0, 1]$
$\bar{p}$	Mean processing time
T^H	Planning horizon
T_a	Average inter-arrival time between two consecutive flights, $\frac{T^H}{N}$

Arrival time and processing time for each flight i were sampled independently from a discrete uniform distribution $h_i \sim U[(i-1)T_a, iT_a]$, $p_i \sim U[60, 100]$. For each flight i, α_{ik} was set to 1 for randomly selected half of the gates and 0 for the other half in the compatibility parameter vector $\boldsymbol{\alpha}_i = (\alpha_{i1}, ..., \alpha_{iM})$. We fixed $T^H = 1000$, $\bar{p} = 80$. N, u are control parameters for the generation of instances. For given N and u, T_a and M are determined by the definition in Table 4.1. $\frac{N\bar{p}}{T^H}$ represents the minimum required number of gates to process all flights within the planning horizon. By dividing the congestion parameter u, the number of gates $M = \lfloor \frac{N\bar{p}}{T^H u} \rfloor$ can be determined. In the artificial data, 5 types of instances were considered, $F100G8$, $F250G20$, $F500G40$ ($u = 1$), $F100G10$ ($u = 0.75$), $F100G16$ ($u = 0.5$), where the numbers after the alphabet F and G represent N and M, respectively. For each instance type, 10 instances were generated.

For the real-world data, we used flight data from 31 days in August 2019 from Atlanta Hartfield-Jackson Airport, which is one of the busiest airports in the world.

The data was obtained from the U.S Bureau of Transportation Statistics website [45]. Each instance was generated based on the data of 1 day and thus, total 31 instances were made for the real-world data. The number of flights per day ranges from 900 to 1300 and the number of gates at Atlanta Hartfield-Jackson Airport is 192. However, the provided data was only for domestic flights and so, we set the number of gates M to 152, which is the number of gates for domestic flights. Since processing times were not explicitly provided, we estimated the values by subtracting the arrival times from the departure times of connecting flights. However, if a processing time was greater than 4 hours, we considered the corresponding flight as two separate flights and sampled the processing times of those two flights in the same way as artificial data. Also, for each flight i, α_{ik} was set to 1 for randomly selected $\lceil \frac{M}{10} \rceil$ gates without replacements and 0 for the other gates in the compatibility parameter vector $\boldsymbol{\alpha}_i = (\alpha_{i1}, ..., \alpha_{iM})$.

All experiments were conducted on Intel Core i7 3.20 GHz processors and 64GB RAM. We used commercial MIP solver Xpress 8.9 [46] to solve LP and MIP problems. All the models and algorithms were implemented in Mosel 5.2 with native interface of C.

4.2 Effects of Algorithmic Parameters of ADP($\mathcal{P}, \eta, \tau$)

In the ADP approach, various parameters such as interpolation ratio η, lookahead horizon τ, and parameters related to the ADP acceleration, were introduced. Since solution quality and computation time of the ADP approach are significantly affected by these parameters, three tests were conducted to quantitatively analyze the effects of parameters. In these tests, all performance measures are reported in average values of 10 instances of the same type, unless explicitly mentioned otherwise.

4.2.1 Sensitivity Analysis on the Value of Interpolation Ratio

Sensitivity analysis on the value of the interpolation ratio η was conducted to find the best value in terms of solution quality. We considered $\eta \in \{0,\ 0.25,\ 0.5,\ 0.75,\ 1\}$. 10 instances from $F100G8$ were used for test instances. The lookahead horizon τ was set to N and no ADP acceleration techniques were used. The Xpress LP Solver algorithm was set to dual simplex which is a default. For performance measures, Gap and $Time$ are used. Gap is a measure of solution quality which is defined by

$$Gap = \frac{ADP.value - z(\mathrm{P})}{ADP.value}\ (\%) \tag{4.1}$$

where $ADP.value$ is the ADP value defined in subsection 3.3.1 and $z(\mathrm{P})$ is the LP relaxation value of the pattern-based model (P). $Time$ is the computation time of the ADP algorithm in seconds. Figure 4.1 shows how Gap and $Time$ change with respect to the value of η.

Among the five levels of η, $\eta = 0$ showed a significantly smaller value of Gap compared to the other levels. In addition, Gap increased as the value of η increased.

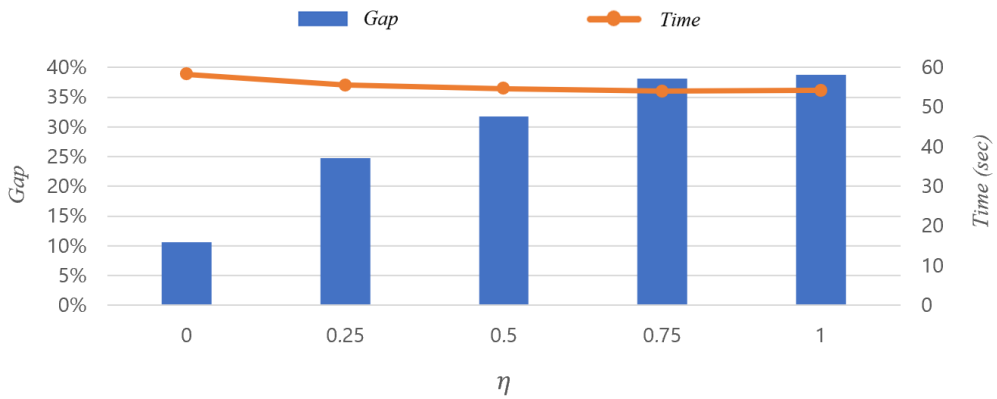


Figure 4.1: Sensitivity analysis on the value of η

This is due to the poor performance of the primal heuristic, the ERD rule. The differences in $Time$ among the five levels of η were negligible, resulting in only a few seconds of variation. Therefore setting the interpolation ratio to 0 emerges as the most attractive strategy. Consequently, we fix $\eta = 0$ for the ADP approach.

4.2.2 Effects of the ADP Acceleration Techniques

In this experiment, we compared the computation time for various combinations of ADP acceleration techniques in section 3.2 to demonstrate the effectiveness of ADP acceleration. We denote `EF(O)` if `EF` is used in the ADP algorithm and `EF(X)` otherwise. Likewise, we denote `EC(O)` and `EC(X)` for `EC` in the same manner. In `RAS`, there are `NAI`, `EAG` and `LAS`. We considered $(\mathtt{A}, \mathtt{B}, \mathtt{C}) \in \{\mathtt{EF(O)}, \mathtt{EF(X)}\} \times \{\mathtt{EC(O)}, \mathtt{EC(X)}\} \times \{\mathtt{LAS}, \mathtt{EAG}, \mathtt{NAI}\}$ where `A`,`B` and `C` represent options for `EF`, `EC` and `RAS`, respectively. Thus, there are total 12 combinations for the ADP acceleration schemes. 10 instances from $F100G8$ were used for test instances. The lookahead horizon τ and early fixing parameter ϵ were set to $N = 100$ and 0, respectively. The Xpress LP Solver algorithm

was set to default. We introduce two performance measures: *Action* and *Iteration*. *Action* represents the number of $\mathrm{MP}_t(a_t)$'s solved for all $a_t \in \mathcal{A}_t$, $t \in F$ in the ADP algorithm. *Iteration* represents the number of iterations in the column generation method of $\mathrm{MP}_t(a_t)$ for all $a_t \in \mathcal{A}_t$, $t \in F$ in the ADP algorithm. Figure 4.2 shows the computation time for all possible ADP acceleration schemes and Figure 4.3 shows *Action* and *Iteration* for all possible ADP acceleration schemes.

All three ADP acceleration techniques significantly reduced the computation time of the ADP algorithm without affecting the solution quality. For `RAS`, `LAS` and `EAG` took much less computation time compared to `NAI`. `LAS` showed slightly better performance compared to `EAG`. In particular, when `EF` was used, the performance deviation among options of `RAS` became bigger. These results can be explained by two factors, *Action* and *Iteration*. `EC` contributed solely to the reduction of *Iteration*, while `EF` was effective in significantly reducing *Action*, which in turn led to a reduction in *Iteration*. When `EF` was not used, `RAS` only reduced *Iteration*, but when `EF` was employed, `RAS` was effective in reducing both *Action* and *Iteration*. This is because evaluations of promising actions first have a bigger potential to provide lower values of approximated value function early in the stage which facilitates `EF` and `EC`.

Additionally, we observed that the effectiveness of ADP acceleration techniques tends to be more prominent, as the problem size increases although we didn't conduct a formal computational experiment for it. In conclusion, (`EF(O)`,`EC(O)`,`LAS`) scheme showed the best performance and based on this result, we fix (`EF(O)`,`EC(O)`,`LAS`) for our ADP acceleration scheme.

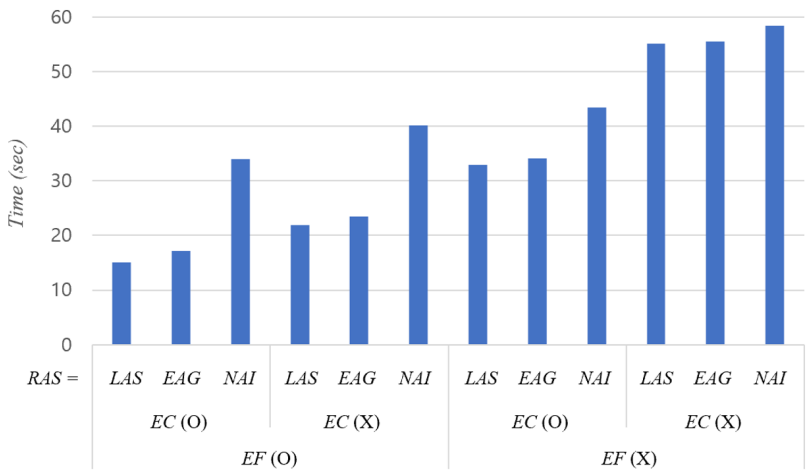


Figure 4.2: Computation time comparison among ADP acceleration schemes

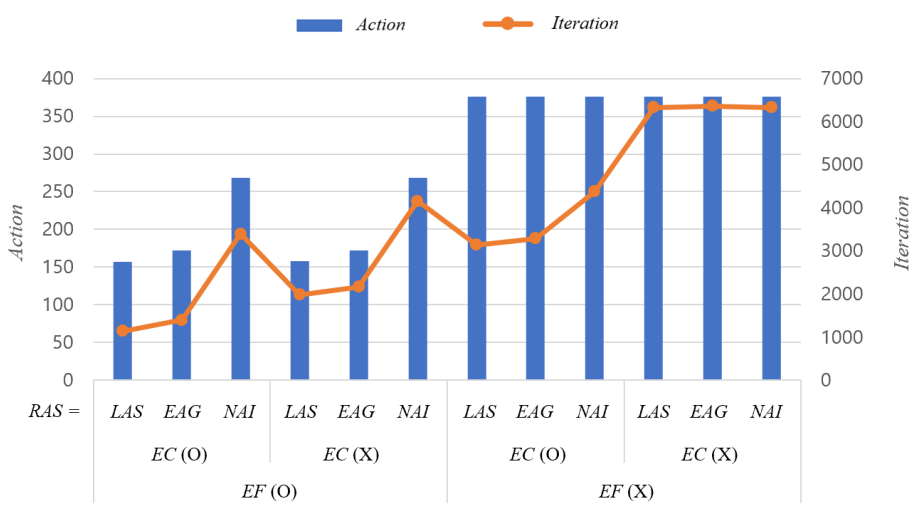


Figure 4.3: Action and iteration comparison among ADP acceleration schemes

Table 4.2: Performance comparison among various ADP parameters for $F500G40$

Parameters		*Time*	*Gap*	*Solved*†
τ	ϵ			
N	0	1801.73	54.8%	0
	0.01	614.44	12.4%	10
	∞	627.48	13.2%	10
Average/Total*		1014.55	26.8%	20
$2M$	0	1800.27	42.9%	0
	0.01	1437.34	27.5%	9
	∞	279.49	26.9%	10
Average/Total*		1172.37	32.4%	19
M	0	907.66	41.0%	10
	0.01	352.12	42.5%	10
	∞	73.07	40.7%	10
Average/Total*		444.29	41.4%	30

† Number of instances solved within time limits
* For *Time* and *Gap*, average values were reported and for *Solved*, total value was reported.

4.2.3 Scalability Test with respect to Parameters τ, ϵ

We tested the ADP approach for various sizes of problems to confirm its scalability. In the test, we controlled two parameters, τ and ϵ, which are concerned with the trade-off between solution quality and computation time. We considered both 3 levels of $\tau \in \{N, 2M, M\}$ and $\epsilon \in \{0, 0.01, \infty\}$. Thus, there are total 9 combinations for the two parameters. We denote a parameter combination by $(\mathtt{A}, \mathtt{B}) \in \{N, 2M, M\} \times \{0, 0.01, \infty\}$ where $\mathtt{A}$ represents level of τ and $\mathtt{B}$ represents level of ϵ. For each instance type, $F100G8$, $F250G20$, $F500G40$, 10 instances were used for test instances. The Xpress LP Solver algorithm was set to Newton-Barrier since it generally performs better than the simplex algorithm for large-scale LP problems. Figure 4.4 and Figure 4.5 show the performance of the ADP approach with various parameter combinations for $F100G8$ and $F250G20$. For $F500G40$, Table 4.2 is presented.

All instances were solved within the time limit for $F100G8$ and $F250G20$. How-

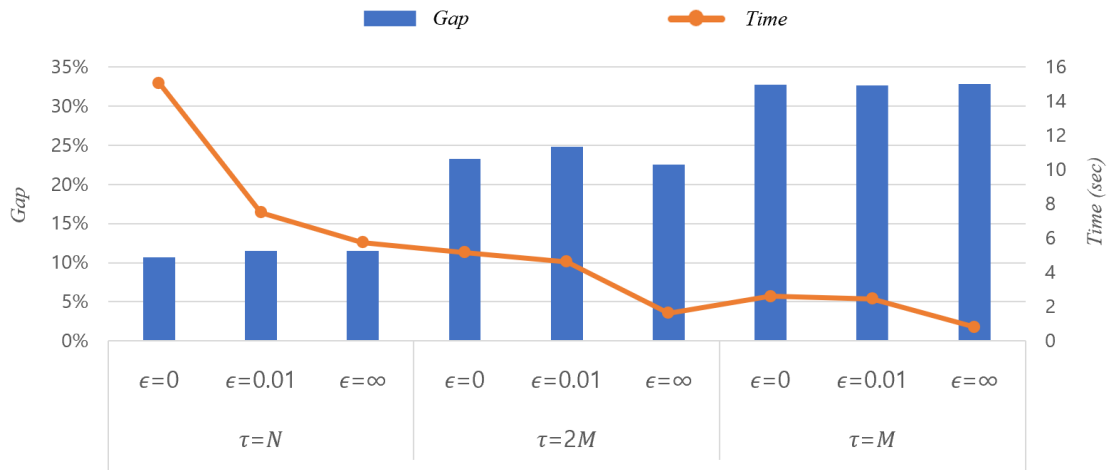


Figure 4.4: Performance comparison among various ADP parameters for $F100G8$

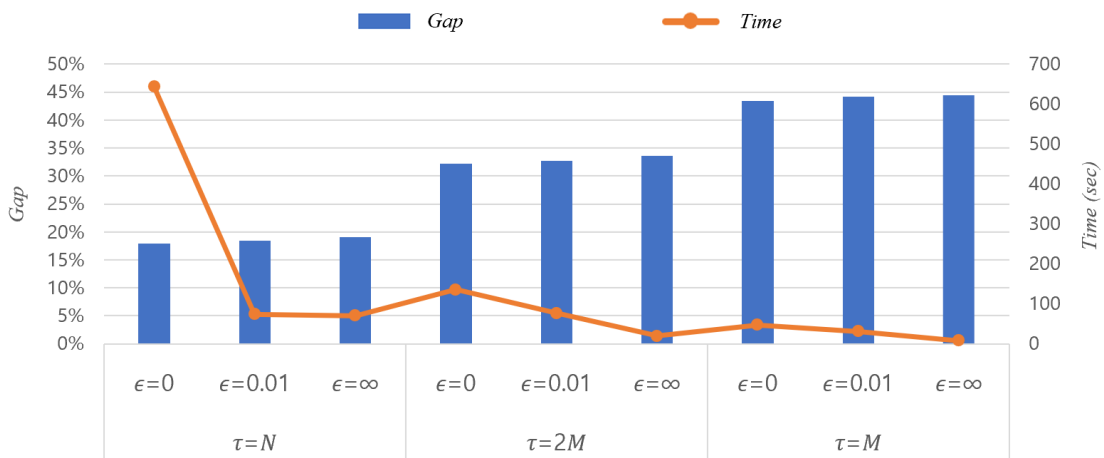


Figure 4.5: Performance comparison among various ADP parameters for $F250G20$

ever, 21 instances out of 90 instances were not solved within the time limit for $F500G40$, especially for large value of τ and small value of ϵ. τ, as expected, greatly influenced both the computation time and solution quality. As τ increased, $Time$ generally got also increased and Gap decreased. However, $(N, 0.01)$ took much shorter computation time than $(2M, 0.01)$ in $F500G40$. This is because of the active column inheritance of $(N, 0.01)$ which significantly reduced the computation time. The effect of ϵ on the solution quality was subtle, due to the `LAS`, while the variation of computation time among ϵ was big. Except for $(N, 0)$, $(2M, 0)$ combination in $F500G40$, Gap differences among the levels of ϵ within the lookahead horizon τ were less than around 2% for all instance types. This implies that it does not significantly deteriorate the solution quality only to explore the first few actions in an action sequence sorted by `LAS` in the ADP approach. Therefore, by adjusting the value of τ and ϵ appropriately, we can apply the ADP approach to the large-scale AGAP efficiently.

4.3 Performance of ADP($\mathcal{P}, \eta, \tau$)

To demonstrate the efficiency of the proposed ADP approach, we tested the ADP approach on both artificial data and real-world data along with other methods. We implemented the following 6 methods:

- **ADP**: ADP algorithm with (`EF(O)`,`EC(O)`,`LAS`) scheme, $\eta = 0$
- **B&P**: Branch-and-Price algorithm of [13]. Multiple column generation strategy was used only for the root node. Initial columns were generated in the same way as the ADP approach. If $(\textit{best integer solution value}) - (\textit{best bound}) < 1$, then the algorithm is terminated since feasible solution values are integral.
- **Solver (B)**: Xpress MIP solver with basic model (B)
- **Solver (N)**: Xpress MIP solver with network model (N)
- **ERD**: Earliest release time dispatching rule
- **EDD**: Earliest due date dispatching rule; **EDD** is same as the **ERD** except that the flights are ordered by the due date, $h_i + p_i$.

ADP, **ERD**, **EDD** are heuristic methods while **B&P**, **Solver (B)**, **Solver (N)** are exact methods.

In experiments for the performance of the ADP approach, we introduce the following performance measures. UB represents the value of a solution given by a certain algorithm at termination. LB represents the best dual bound of a certain algorithm at termination if provided. $Node$ represents the number of **B&P** nodes searched within the time limit. All performance measures were reported on individual instances.

4.3.1 Test on Artificial Data

Congestion is the scale of difficulty for instances because instances with a higher value of congestion are more prone to confront delay propagation. To test the performance of various methods on instances with diverse congestion, we used 3 instance types $F100G16$ ($u = 0.5$), $F100G10$ ($u = 0.75$), $F100G8$ ($u = 1$). For each instance type, $F100G16$, $F100G10$, $F100G8$, 10 instances were used for test instances. In **ADP**, the lookahead horizon τ and `EF` parameter ϵ were set to N and 0, repectively. The Xpress LP solver algorithm was set to default. Table 4.3 shows the performance of various methods on artificial data.

For $F100G16$, all methods found an optimal solution for all 10 instances. Except for **Solver (N)**, all methods terminated in less than a second. For $F100G10$, only **ADP** and **B&P** found optimal solutions for all 10 instances. But the computation time differed by around 100 times. **Solver (B)** and **Solver (N)** found optimal solutions for 7 instances and 4 instances, respectively. For two dispatching rules, none of the methods found any optimal solution. For $F100G8$, **ADP** outperformed all other methods significantly in terms of solution quality. All exact methods showed poor performance for high congestion instances. In $F100G8$, UB of exact methods were similar to that of **ERD** and **EDD**. However, the computation time of exact methods reached the time limit while **ERD** and **EDD** found solutions in less than a second. For two dispatching rules, no one method dominated the other method. But **ERD** showed better UB than **EDD** for most instances. One interesting observation is that the $Node$ of **B&P** was all 2 in $F100G8$. This is because solving subproblems took most of the computation time.

Table 4.3: Performance comparison of various methods on artificial data

Instance		**ADP**		**B&P**				**Solver (B)**			**Solver (N)**			**ERD**†	**EDD**†
util.	*inst.#*	*Time*	*UB*	*Time*	*UB*	*LB*	*Node*	*Time*	*UB*	*LB*	*Time*	*UB*	*LB*	*UB*	*UB*
0.5	1	0.00	0	0.00	0	0.00	1	0.74	0	0.00	11.22	0	0.00	0	0
	2	0.00	0	0.00	0	0.00	1	0.64	0	0.00	23.32	0	0.00	0	0
	3	0.00	0	0.00	0	0.00	1	0.66	0	0.00	27.52	0	0.00	0	0
	4	0.00	0	0.00	0	0.00	1	0.58	0	0.00	18.01	0	0.00	0	0
	5	0.00	0	0.00	0	0.00	1	0.59	0	0.00	19.66	0	0.00	0	0
	6	0.00	0	0.00	0	0.00	1	0.67	0	0.00	10.00	0	0.00	0	0
	7	0.00	0	0.00	0	0.00	1	0.59	0	0.00	29.70	0	0.00	0	0
	8	0.00	0	0.00	0	0.00	1	0.78	0	0.00	9.64	0	0.00	0	0
	9	0.00	0	0.00	0	0.00	1	0.89	0	0.00	21.54	0	0.00	0	0
	10	0.00	0	0.00	0	0.00	1	0.58	0	0.00	25.38	0	0.00	0	0
	Average	0.00	0.00	0.00	0	0.00	1.00	0.67	0.00	0.00	19.60	0.00	0.00	0.00	0.00
0.75	1	3.07	1	175.28	1	1.00	39	1,000.31	1	1.00	1,726.62	1	1.00	208	288
	2	2.62	0	134.54	0	0.00	41	197.02	0	0.00	1,169.29	0	0.00	143	314
	3	2.96	0	208.34	0	0.00	30	130.13	0	0.00	1,763.57	0	0.00	154	175
	4	3.16	0	330.36	0	0.00	52	967.43	0	0.00	1,800.18*	70	0.00	161	251
	5	2.77	0	80.44	0	0.00	27	1,008.94	0	0.00	1,800.13*	373	0.00	153	226
	6	4.45	4	1211.01	4	3.40	81	1,800.11*	115	0.00	1,800.10*	386	0.00	267	322
	7	2.68	0	230.88	0	0.00	68	106.97	0	0.00	1,800.05*	47	0.00	121	261
	8	4.16	3	115.49	3	3.00	12	1,800.06*	137	0.00	1,800.09*	819	0.00	340	297
	9	2.08	0	163.63	0	0.00	61	43.64	0	0.00	438.69	0	0.00	116	171
	10	3.51	17	335.13	17	17.00	24	1,800.06*	86	0.00	1,800.07*	312	0.00	255	208
	Average	3.15	2.50	298.51	2.50	2.44	43.50	885.47	33.90	0.10	1,589.88	200.80	0.10	191.80	251.30
1	1	17.83	1,015	1,815.82*	2,116	891.05	2	1,800.18*	2,147	0.00	1,800.14*	2,075	102.09	2,116	2,205
	2	15.89	974	1,817.61*	2,067	863.73	2	1,800.10*	2,203	0.00	1,800.11*	2,944	0.00	2,067	2,082
	3	18.38	803	1,811.31*	1,563	664.57	2	1,800.08*	2,055	0.00	1,800.26*	1,497	94.53	1,563	1,621
	4	19.21	816	1,807.64*	1,629	680.02	2	1,800.13*	2,030	0.00	1,800.09*	1,633	100.93	1,629	2,194
	5	12.86	1,532	1,821.30*	2,990	1,458.17	2	1,800.07*	2,905	0.00	1,800.07*	2,457	122.18	2,990	2,683
	6	14.01	555	1,811.26*	1,085	534.35	2	1,800.07*	1,786	0.00	1,800.07*	1,169	105.64	1,085	1,406
	7	18.61	1,044	1,815.35*	2,074	940.21	2	1,800.18*	2,291	0.00	1,800.06*	2,012	112.38	2,074	2,324
	8	10.29	1,191	1,815.48*	2,518	1,123.30	2	1,800.20*	2,658	0.00	1,800.77*	2,007	134.10	2,518	3,117
	9	15.46	1,499	1,819.46*	2,462	1,308.12	2	1,800.06*	2,665	0.00	1,800.21*	2,607	126.68	2,462	2,869
	10	18.51	921	1,819.02*	1,610	808.75	2	1,800.06*	2,223	0.00	1,800.58*	1,960	91.24	1,610	2,048
	Average	16.10	1,035.00	1,815.42	2,011.40	927.23	2.00	1,800.11	2,296.30	0.00	1,800.23	2,036.10	98.98	2,011.40	2,254.90

† Methods whose solutions were found within 0.01 second

* Instances where the algorithm reached the time limit

4.3.2 Test on Real-world Data

Real-world instances have relatively low congestion but sparse compatibility between flights and gates. In real-world data, 152 gates and more than 1000 flights are considered. The purpose of this experiment is to demonstrate that the proposed ADP approach can efficiently solve the AGAP in real-world airports by testing on large-scale, real-world instances along with other existing methods. In **ADP**, the lookahead horizon τ and `EF` parameter ϵ were set to M and ∞, respectively. The Xpress LP solver algorithm was set to Newton-Barrier. Table 4.4 shows the performance of various methods on real-world data. In **Instance** column, the number next to f, g, day represents the number of flights, gates, day of month, respectively.

Among the 31 instances, 29 instances have 0 optimal delays, which can be ascertained from the UB of **ADP** and LB of **B&P**. For 29 instances with 0 optimal delays, **ADP** found optimal solutions for all instances while **B&P** and **Solver (B)** found 5 and 22 optimal solutions, respectively. For the other 2 instances, the absolute gap of (UB of **ADP**) $-$ (LB of **B&P**) were 2 and 8. In terms of solution quality, **ADP** showed the best performance of all methods for all 31 instances. In addition, the average computation time of **ADP** was less than 5 minutes, which is a significantly smaller value compared to other exact methods. **Solver (N)** reaches the time limit for all instances and UB varies from a hundred thousand to a million. The poor performance of **Solver (N)** is due to the large number of variables of network model (N). One noticeable result is that the average UB of **EDD** was over 40 times greater than the average UB of **ERD**, where two methods had a similar scale of values in test on artificial data. The performance gap of **ERD** and **EDD** between the two datasets comes from the sparsity of compatibility in instances.

Table 4.4: Performance comparison of various methods on real-world data

Instance	ADP		B&P				Solver (B)			Solver (N)			ERD†	EDD†
	Time	*UB*	*Time*	*UB*	*LB*	*Node*	*Time*	*UB*	*LB*	*Time*	*UB*	*LB*	*UB*	*UB*
f1439g152day1	319.82	0	1,800.84	38	0	59	674.68	0	0	1,826.29	935,167	0	38	5,011
f1426g152day2	148.52	0	1,803.82	11	0	102	619.67	0	0	1,826.03	925,572	0	11	3,447
f1159g152day3	145.41	0	686.11	0	0	113	377.07	0	0	1,815.03	145,013	0	32	2,632
f1377g152day4	123.24	0	1,033.65	0	0	94	632.56	0	0	1,824.24	707,034	0	55	3,220
f1425g152day5	351.77	0	1,810.76	105	0	117	701.08	0	0	1,823.63	620,983	0	105	3,804
f1423g152day6	215.79	0	1,803.31	131	0	126	1,922.62	59	0	1,824.19	803,671	0	131	4,543
f1450g152day7	1,772.38	0	1,800.91	101	0	74	829.70	0	0	1,826.93	753,934	0	101	5,195
f1451g152day8	363.87	0	1,824.61	214	0	70	1,925.90	220	0	1,825.00	793,745	0	214	4,689
f1435g152day9	212.21	0	1,802.23	50	0	71	733.17	0	0	1,824.16	742,758	0	50	3,242
f1157g152day10	160.79	0	1,499.61	0	0	232	421.08	0	0	1,815.00	357,964	0	61	3,116
f1367g152day11	89.38	0	1,801.90	17	0	86	544.85	0	0	1,824.10	589,434	0	17	2,606
f1411g152day12	162.85	0	1,807.16	60	0	125	647.84	0	0	1,826.00	802,358	0	60	2,671
f1380g152day13	228.70	0	1,802.98	87	0	143	620.27	0	0	1,823.54	699,677	0	87	3,832
f1400g152day14	280.56	0	1,691.03	0	0	79	1,904.93	18	0	1,823.53	825,448	0	119	4,634
f1396g152day15	167.54	0	1,807.09	48	0	81	639.29	0	0	1,823.67	703,773	0	48	2,759
f1405g152day16	261.71	0	1,802.72	119	0	149	676.31	0	0	1,823.67	653,289	0	119	2,592
f1108g152day17	150.75	3	1,800.01	99	0	161	1,872.32	31	0	1,813.59	572,105	0	99	3,449
f1354g152day18	205.24	0	1,804.28	39	0	75	570.61	0	0	1,820.45	787,674	0	39	4,140
f1402g152day19	283.54	0	1,810.23	145	0	74	1,912.21	121	0	1,823.72	669,166	0	145	3,727
f1384g152day20	254.05	0	1,802.59	98	0	105	654.13	0	0	1,824.74	679,747	0	98	5,058
f1368g152day21	270.05	0	1,810.96	57	0	74	626.72	0	0	1,823.53	706,099	0	57	4,936
f1382g152day22	198.34	0	1,810.46	83	0	105	601.27	0	0	1,823.31	617,817	0	83	4,051
f1376g152day23	192.35	0	1,801.06	88	0	129	642.48	0	0	1,822.07	821,536	0	88	3,467
f1092g152day24	202.45	78	1,809.40	284	76	124	1,867.43	139	0	1,813.00	564,879	0	284	4,440
f1325g152day25	187.81	0	1,800.00	47	0	95	552.10	0	0	1,823.41	665,936	0	47	3,558
f1367g152day26	233.69	0	1,801.66	73	0	176	580.33	0	0	1,822.44	709,232	0	73	2,469
f1360g152day27	262.90	0	1,232.43	0	0	56	1,900.74	65	0	1,824.01	729,153	0	116	4,793
f1361g152day28	232.16	0	1,813.82	49	0	71	593.89	0	0	1,824.38	778,406	0	49	3,200
f1371g152day29	217.26	0	1,809.00	45	0	119	570.58	0	0	1,825.68	697,398	0	45	2,438
f1398g152day30	278.17	0	1,822.50	173	0	82	1,911.97	154	0	1,824.75	846,428	0	173	3,306
f946g152day31	125.60	30	1,800.27	147	22	314	1,856.60	65	0	1,809.39	663,727	0	147	4,754
Average	267.71	3.58	1,713.14	77.68	3.16	112.29	986.59	28.13	0.00	1,822.37	695,778.16	0.00	90.03	3,734.81

† Methods whose solutions were found within 1 second

* Instances where the algorithm reached the time limit

Chapter 5

Conclusion

In this thesis, we proposed a scalable optimization-based ADP approach for AGAP to efficiently handle the large-scale nature of real-world airports. In the ADP approach, the value function approximation with consideration of a lookahead horizon was utilized. By comparing the LP relaxation value of various IP models for AGAP, the pattern-based model, which gives the strongest bound, was used for approximating the value function. However, the pattern-based model necessitates excessive computation burden arising from a large number of column generation iterations and action evaluations. To overcome this computational issue, we developed an efficient column generation method and ADP acceleration techniques.

Through computational experiments, we demonstrated the scalability of the ADP approach and its applicability to real-world airports in practice. One strength of the proposed ADP approach is that we can control the trade-off between solution quality and computation time by adjusting algorithmic parameters depending on specific situations. For example, when establishing in-advance planning, we can put more weight on solution quality by considering a longer lookahead horizon and smaller value of `EF` threshold. In contrast, when establishing adaptive replanning, we can put more weight on computation time by considering a shorter lookahead

horizon and larger value of `EF` threshold.

For future research directions, reflecting realistic restrictions can be considered. The AGAP in this study only considered adjacency and compatibility constraints. However, in real-world airports, allocating flights to gates is coupled with other material and human resources so AGAP needs to be considered in a more broad aspect. In addition, uncertainty in arrival time and processing can also be considered. Due to the complex airport operation system and growing arrival delays, changes in flight schedules are common. Reinforcement learning and stochastic programming can be used for AGAP under uncertainty. Finally, a generalization of the proposed ADP approach to PMSP can be made as AGAP and PMSP share a similar problem structure.

Bibliography

[1] DELMIA, "Airport operations." https://www.3ds.com/products-services/delmia/solutions/aerospace-defense/airport-operations/, 2023. Accessed: 2023-05-15.

[2] DAIFUKU, "Airport operation system (aos)." https://daifukuatec.com/airport-technologies/airport-operation-systems, 2023. Accessed: 2023-05-15.

[3] S. Yan and C.-H. Tang, "A heuristic approach for airport gate assignments for stochastic flight delays," *European Journal of Operational Research*, vol. 180, pp. 547–567, 2007.

[4] D. Zhang, H. H. Lau, and C. Yu, "A two stage heuristic algorithm for the integrated aircraft and crew schedule recovery problems," *Computers & Industrial Engineering*, vol. 87, no. 1, pp. 436–453, 2015.

[5] L. C. Kim Y., Kim J., *A Study on the Development and Application of Gate Assignment Algorithm.* Korea Transport Institute, 2001.

[6] D. Zhang and D. Klabjan, "Optimization for gate re-assignment," *Transportation Research Part B: Methodological*, vol. 95, pp. 260–284, 2017.

[7] U. Dorndorf, A. Drexl, Y. Nikulin, and E. Pesch, "Flight gate scheduling: State-of-the-art and recent developments," *Omega*, vol. 35, no. 3, pp. 326–334, 2007.

[8] G. S. Daş, F. Gzara, and T. Stützle, "A review on airport gate assignment problems: Single versus multi objective approaches," *Omega*, vol. 92, p. 102146, 2020.

[9] J. Kim, B. Goo, Y. Roh, C. Lee, and K. Lee, "A branch-and-price approach for airport gate assignment problem with chance constraints," *Transportation Research Part B: Methodological*, vol. 168, pp. 1–26, 2023.

[10] E. B. Peterson, K. Neels, N. Barczi, and T. Graham, "The economic cost of airline flight delay," *Journal of Transport Economics and Policy (JTEP)*, vol. 47, no. 1, pp. 107–121, 2013.

[11] J. Calzada and X. Fageda, "Airport dominance, route network design and flight delays," *Transportation Research Part E: Logistics and Transportation Review*, vol. 170, p. 103000, 2023.

[12] I. Kaliszewski, J. Miroforidis, and J. Stańczak, "The airport gate assignment problem–multi-objective optimization versus evolutionary multi-objective optimization," *Computer Science*, vol. 18, pp. 41–52, 2017.

[13] Y. Li, J.-P. Clarke, and S. S. Dey, "Using submodularity within column generation to solve the flight-to-gate assignment problem," *Transportation Research Part C: Emerging Technologies*, vol. 129, p. 103217, 2021.

[14] B. Maharjan and T. I. Matis, "Multi-commodity flow network model of the flight gate assignment problem," *Computers & Industrial Engineering*, vol. 63, no. 4, pp. 1135–1144, 2012.

[15] C.-H. Tang and W.-C. Wang, "Airport gate assignments for airline-specific gates," *Journal of Air Transport Management*, vol. 30, pp. 10–16, 2013.

[16] J. Bi, F. Wang, C. Ding, D. Xie, and X. Zhao, "The airport gate assignment problem: A branch-and-price approach for improving utilization of jetways," *Computers & Industrial Engineering*, vol. 164, p. 107878, 2022.

[17] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, vol. 6. Springer, 2016.

[18] Z.-L. Chen and W. B. Powell, "A column generation based decomposition algorithm for a parallel machine just-in-time scheduling problem," *European Journal of Operational Research*, vol. 116, no. 1, pp. 220–232, 1999.

[19] A. P. Vepsalainen and T. E. Morton, "Priority rules for job shops with weighted tardiness costs," *Management science*, vol. 33, no. 8, pp. 1035–1047, 1987.

[20] L. Yang-Kuei and L. Chi-Wei, "Dispatching rules for unrelated parallel machine scheduling with release dates," *The International Journal of Advanced Manufacturing Technology*, vol. 67, pp. 269–279, 2013.

[21] C.-H. Lee, "A dispatching rule and a random iterated greedy metaheuristic for identical parallel machine scheduling to minimize total tardiness," *International Journal of Production Research*, vol. 56, no. 6, pp. 2292–2308, 2018.

[22] X. Liu, F. Chu, F. Zheng, C. Chu, and M. Liu, "Parallel machine scheduling with stochastic release times and processing times," *International Journal of Production Research*, vol. 59, no. 20, pp. 6327–6346, 2021.

[23] J. A. Bennell, M. Mesgarpour, and C. N. Potts, "Airport runway scheduling," *Annals of Operations Research*, vol. 204, pp. 249–270, 2013.

[24] A. Lieder, D. Briskorn, and R. Stolletz, "A dynamic programming approach for the aircraft landing problem with aircraft classes," *European Journal of Operational Research*, vol. 243, no. 1, pp. 61–69, 2015.

[25] D. Briskorn and R. Stolletz, "Aircraft landing problems with aircraft classes," *Journal of Scheduling*, vol. 17, pp. 31–45, 2014.

[26] A. Salehipour, M. Modarres, and L. M. Naeni, "An efficient hybrid metaheuristic for aircraft landing problem," *Computers & Operations Research*, vol. 40, no. 1, pp. 207–213, 2013.

[27] N. R. Sabar and G. Kendall, "An iterated local search with multiple perturbation operators and time varying perturbation strength for the aircraft landing problem," *Omega*, vol. 56, pp. 88–98, 2015.

[28] H. Pinol and J. E. Beasley, "Scatter search and bionomic algorithms for the aircraft landing problem," *European Journal of Operational Research*, vol. 171, no. 2, pp. 439–462, 2006.

[29] A. Faye, "Solving the aircraft landing problem with time discretization approach," *European Journal of Operational Research*, vol. 242, no. 3, pp. 1028–1038, 2015.

[30] W. B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality.* John Wiley & Sons, 2011.

[31] W. B. Powell, "What you should know about approximate dynamic programming," *Naval Research Logistics (NRL)*, vol. 56, no. 3, pp. 239–249, 2009.

[32] W. B. Powell, "A unified framework for stochastic optimization," *European Journal of Operational Research*, vol. 275, no. 3, pp. 795–821, 2019.

[33] Q. Deng and B. F. Santos, "Lookahead approximate dynamic programming for stochastic aircraft maintenance check scheduling optimization," *European Journal of Operational Research*, vol. 299, no. 3, pp. 814–833, 2022.

[34] J. Brinkmann, M. W. Ulmer, and D. C. Mattfeld, "Dynamic lookahead policies for stochastic-dynamic inventory routing in bike sharing systems," *Computers & Operations Research*, vol. 106, pp. 260–279, 2019.

[35] W. B. Powell, "Designing lookahead policies for sequential decision problems in transportation and logistics," *IEEE Open Journal of Intelligent Transportation Systems*, vol. 3, pp. 313–327, 2022.

[36] J. Fang, L. Zhao, J. C. Fransoo, and T. Van Woensel, "Sourcing strategies in supply risk management: An approximate dynamic programming approach," *Computers & Operations Research*, vol. 40, no. 5, pp. 1371–1382, 2013.

[37] M. Heydar, E. Mardaneh, and R. Loxton, "Approximate dynamic programming for an energy-efficient parallel machine scheduling problem," *European Journal of Operational Research*, vol. 302, no. 1, pp. 363–380, 2022.

[38] D. Bertsekas and J. N. Tsitsiklis, *Neuro-dynamic programming.* Athena Scientific, 1996.

[39] Y. Lee and K. Lee, "New integer optimization models and an approximate dynamic programming algorithm for the lot-sizing and scheduling problem with sequence-dependent setups," *European Journal of Operational Research*, vol. 302, no. 1, pp. 230–243, 2022.

[40] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. Savelsbergh, and P. H. Vance, "Branch-and-price: Column generation for solving huge integer programs," *Operations research*, vol. 46, no. 3, pp. 316–329, 1998.

[41] G. Sierksma and G. A. Tijssen, "Routing helicopters for crew exchanges on off-shore locations," *Annals of Operations Research*, vol. 76, no. 0, pp. 261–286, 1998.

[42] S. Beraudy, N. Absi, and S. Dauzère-Pérès, "Timed route approaches for large multi-product multi-step capacitated production planning problems," *European Journal of Operational Research*, vol. 300, no. 2, pp. 602–614, 2022.

[43] D. Bertsimas and J. N. Tsitsiklis, *Introduction to linear optimization*, vol. 6. Athena scientific Belmont, MA, 1997.

[44] N. Buchbinder, M. Feldman, J. Seffi, and R. Schwartz, "A tight linear time (1/2)-approximation for unconstrained submodular maximization," *SIAM Journal on Computing*, vol. 44, no. 5, pp. 1384–1402, 2015.

[45] BTS, "Airline on-time performance data." https://transtats.bts.gov/Tables.asp?QO_VQ=EFD&QO_anzr=Nv4yv0r%FDb0-gvzr%FDcr4s14zn0pr%FDQn6n&QO_fu146_anzr=b0-gvzr, 2023. Accessed: 2023-02-21.

[46] Xpress, "Xpress 8.9." http://www.fico.com/en, 2016.

[47] X.-B. Hu and E. Di Paolo, "An efficient genetic algorithm with uniform crossover for the multi-objective airport gate assignment problem," in *2007 IEEE Congress on Evolutionary Computation*, pp. 55–62, IEEE, 2007.

[48] M. Jayamohan and C. Rajendran, "New dispatching rules for shop scheduling: a step forward," *International Journal of Production Research*, vol. 38, no. 3, pp. 563–586, 2000.

[49] D. Bertsimas and R. Demir, "An approximate dynamic programming approach to multidimensional knapsack problems," *Management Science*, vol. 48, no. 4, pp. 550–565, 2002.

[50] IATA, "Air travel growth continues in february." https://www.iata.org/en/pressroom/2023-releases/2023-04-04-02/, 2023. Accessed: 2023-04-10.

국문초록

본 연구에서는 공항 운영상의 제약을 고려하여 항공기를 게이트에 적절히 배분하는 공항 게이트 할당 문제를 다룬다. 현실에서는 많은 수의 항공기와 게이트를 고려해야 하고, 동시에 상황 변화에 따른 계획의 재수립이 빈번하여, 이에 유연하고 신속하게 대처할 수 있는 효율적인 방법이 필요하다. 이를 위해 본 연구에서는 전방 탐색 길이를 고려한 근사동적계획 해법을 제안한다. 근사동적계획법에서 가치함수를 추정하기 위해 패턴기반모형의 선형계획완화문제의 하한을 활용한다. 패턴기반모형은 강한 하한을 제공하나 지수적으로 많은 개수의 변수를 가지고 있어 이 모형을 활용할 시 매우 큰 계산적 부담이 발생한다. 이 문제점을 극복하기 위해 효율적인 열생성 방법 및 근사동적계획 가속화 기법을 제안한다. 다양한 실험을 통해 열생성 방법 및 근사동적계획 가속화 기법의 효과성을 확인하였다. 그리고 실제 데이터에 기반한 실험을 통하여 제안한 해법이 기존 방법 대비 합리적인 시간 내에 좋은 성능의 해를 제공하는 것을 확인하였다.

주요어: 공항 게이트 할당 문제, 열생성기법, 근사동적계획법, 가속화 기법, 확장수리모형

학번: 2021-26136

감사의 글

연구실에서 생활하는 동안 교수님의 지도 및 연구실 선후배님들의 도움으로 개인적으로 많은 성장을 할 수 있었습니다. 그 결과 무사히 석사 논문을 작성하고 학위 과정을 마무리할 수 있었습니다.

먼저 바쁘신 와중에도 시간을 내어 심사를 맡아주신 심사위원장 홍성필 교수님과 심사위원 문일경 교수님께 감사의 말씀을 전합니다. 그리고 저의 지도교수님이신 이경식 교수님께 깊은 존경과 감사의 말씀을 드립니다. 한없이 부족한 저에게 지난 석사 과정 동안 학문적인 가르침과 더불어 진중한 삶에 대한 방향성에 대해 많은 시간을 할애하여 조언을 해주셨습니다. 학교에서 공부 및 연구에 전념할 수 있었던 것을 사회에 환원할 수 있는 사람이 될 수 있도록 노력하겠습니다.

연구실 선후배님들께도 감사의 말씀을 전합니다. 연구실 식구의 일을 자기 일처럼 생각하며 서로를 돕고 이끌었던 동료들 덕분에 즐겁게 생활할 수 있었습니다. 연구실에서 맺은 인연이 앞으로도 이어졌으면 좋겠습니다. 마지막으로 제 곁에서 늘 응원해주신 저의 가족에게 감사합니다. 아버지, 어머니, 누나 모두 건강하고 활기차게 지내시길 바랍니다.