ABSTRACT

BAGHDADI, MAJED ABUBAKR M. Practical Approaches for Solving the Runway Scheduling Problem – The Departure Case. (Under the direction of Dr. Thom J. Hodgson and Dr. Russell E. King.)

The Runway Scheduling Problem (RSP) is common at large airports. With increased demand for flights over time, the capacity of runways may become insufficient when running in typical fashion: First-Come-First-Served (FCFS). There are issues that make the RSP challenging, such as the required separation time between different aircraft types, airport layout, and the dynamic nature of the problem. It is important to solve the problem for multiple objectives since there are multiple stakeholders, some with conflicting interests.

The goal of this dissertation is to provide efficient practical approaches to solve the RSP for departing aircraft in different operational conditions. There are many operational conditions that exist. The first condition is: 1 runway, 2 aircraft types. A constructive method is developed to minimize completion time (C_{max}) or minimize maximum individual delay (F_{max}). An extension to solve large problems dynamically is proposed. A computational experiment using New York JFK Airport data is performed to test the approach against FCFS. Better results are found for C_{max} and F_{max} than using FCFS.

The second condition is: 1 runway, 3 types. Many airports deal with two types of aircraft most of the day. However, at certain times, they operate with three aircraft types. An approach to solve the problem for C_{max} or F_{max} for small instances is introduced. An extension to solve large cases dynamically is proposed. A computational experiment using New York JFK Airport data is performed to test the approach against FCFS. Better results are found for C_{max} and F_{max} than using FCFS.

The third condition is: 2 runways, 2 types. When two runways are available, assignment of aircraft to runways is added to the sequencing problem. A proposed approach based on dividing the problem into two independent sub-problems is introduced, followed by a dynamic approach for large instances. A computational experiment using Atlanta Hartsfield Airport data is performed to test the approach against FCFS. Better results are found for C_{max} and F_{max} than using FCFS.

Practical Approaches for Solving the Runway Scheduling Problem – The Departure Case

by
Majed A. Baghdadi

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Industrial and Systems Engineering

Raleigh, North Carolina

2018

APPROVED BY:

_____	_____
Dr. Thom J. Hodgson	Dr. Russell E. King
Committee Co-Chair	Committee Co-Chair
_____	_____
Dr. Donald P. Warsing	Dr. Michael G. Kay

DEDICATION

To My Late Mom, Huda. May God Bless You

My Wife, Sendyan

And

My Two Beautiful Children, Yusuf and Maryam

اللهُمَ لكَ الحَمْدُ حَمْداً يَلِيقُ بِجَلالِكَ وَعَظَمَتِك

BIOGRAPHY

Majed Abubakr M Baghdadi was born in Jeddah, Saudi Arabia in May 28th, 1981. He attended high school at Rawdat Al-Ma'aref Private School in Jeddah. He earned his Bachelor degree in Electrical and Computer Engineering in 2005 from King Abdulaziz University in Jeddah. He worked at King Abdulaziz University as a teacher assistant in the industrial engineering department from 2005 to 2007. In 2007, he received a scholarship from the university to pursue graduate education abroad. He earned a Master degree in Management Science in 2010 from University of Waterloo in Canada. He joined North Carolina State University for a doctoral degree in industrial engineering in 2011. He plans to return to work at King Abdulaziz University after completing his PhD degree as an assistant professor in the industrial engineering department.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Thom Hodgson, for guiding me through the research process, for teaching me to NEVER GIVE UP and for believing on me. I would also like to thank the rest of the committee, Dr. King, Dr. Warsing, and Dr. Kay for their valuable feedback and editing of the dissertation.

I want to thank my undergraduate school and sponsor, King Abdulaziz University for the graduate studies scholarship award. My special thanks to Dr. Abdullah Bafail, Dr. Siraj Abed, Dr. Hani Aburas, Dr. Mohammad Subaih, and my undergraduate advisor, Dr. Bahattin Karagozoglu.

There are many friends and supporting people who helped me finishing this project. Specially, I would like to thank Abdullah Bukhary, Justin, Yunis, Mary Njaramba, Yuka, Jullie, Mosab AlAmoody, Mohammad Arafa, Anmar Kinsara, Rodrigo, Azmat, Caglar, Melisa, Ashraf, Brandon, Mac McNair, and P.J. Adams. Thanks to many others who I did not mention their names here.

Finally, special thanks to my wife, Sendyan, for her unbelievable patience and support throughout this journey, my father and late mother for raising me and teaching me valuable lessons in life, and my siblings, Mohammad and Nada, for their continuous support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1 – Introduction

The runway scheduling problem (RSP) is a common problem at large airports. Airplanes queue accumulate quickly for both takeoff and landing operations with limited runway capacity. Delays occur and cause unwanted operational problems and unnecessary costs. The main responsible party for controlling runway operations is the airport traffic control tower (ATC). With many responsibilities at hand it is necessary to be able to operate as safe as possible as a first priority, then other considerations can be dealt with such as operating with minimum cost and/or maximum efficiency.

With the increased demand for flights over time, the capacity of many runways becomes insufficient when running in a simple fashion such as first-come-first-served. It has become important to use smart techniques to run the take-off and landing process more efficiently. The problem of operating the runway efficiently is challenging and there are many issues that cause this. One reason is the structure of the problem. The runway is the bottleneck of the airport system (Idris et al., 1998). Many aircraft are ready to take off and many are ready to land at the same time but the runway is a limited resource for handling these operations. Also, there are strict operational standards enforced by the International Civil Aviation Organization (ICAO) and the Federal Aviation Agency (FAA) to insure safety and other issues. One such key standard is the time and distance separation requirement between aircraft using the same runway. This varies between consecutive aircraft based on their size and type of operation. Because of the sequence dependency involved, this makes the aircraft sequencing problem (landing or taking off) an NP-hard problem (Bennell et al., 2013)

Another reason that makes the problem challenging is the time required to solve the problem which must be very fast (near real-time) due to the nature of the problem (new events occurring every minute). Even with the high technology resources available these days that might be used to solve the problem, it might take a relatively long time to provide efficient solutions.

There have been many efforts in the literature to solve the airport runway scheduling problem (RSP), both on the landing (aircraft landing problem, ALP) and the take-off (aircraft takeoff problem, ATP), and mixed mode operations. Many models have been considered; such as

mixed integer programming (MIP), machine scheduling, the traveling sales man (TSP), and queueing. Also, many solution methodologies have been considered such as dynamic programming, branch and bound, heuristics and meta-heuristics (detailed literature review is in Chapter 2). There is still room for improvement especially in the area of providing fast and efficient solutions that can be implemented in practical environments.

There are several types of airport hubs in the United States according to the FAA. They are Large, Medium, Small and Non-hub. There are 30 large hubs with passenger traffic ranging from 16 million to 96 million passengers per year. Active runways range from one to five (most commonly parallel), and each has a different geometrical configuration. Three examples of large airports are Hartsfield Jackson Atlanta International Airport (ATL) in Atlanta, GA (Figure 1), Charlotte Douglas International Airport (CLT) in Charlotte, NC (Figure 2) and New York JFK International Airport (JFK) in New York (Figure 3). See Appendix A for details.

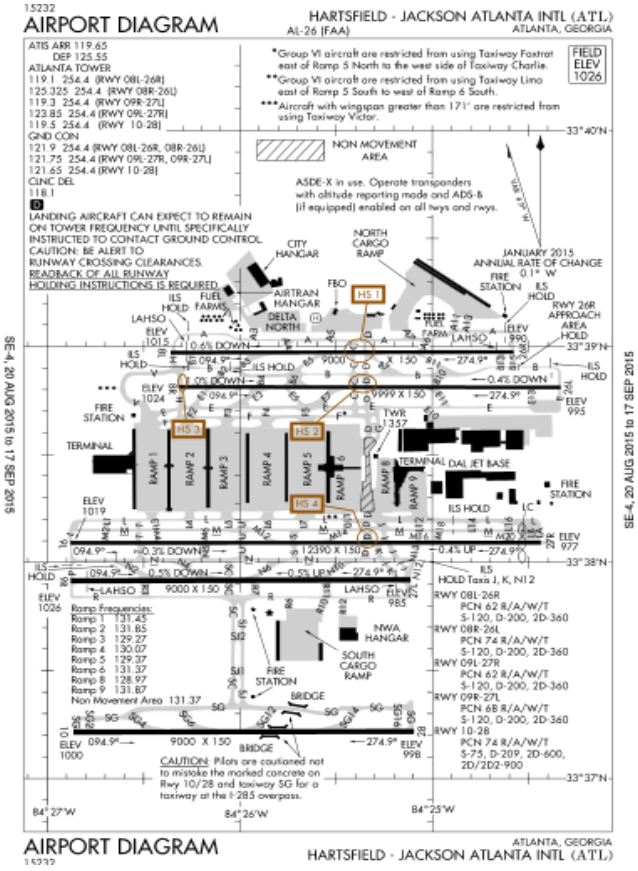

Figure 1: Hartsfield Jackson Atlanta International (ATL) Airport Diagram

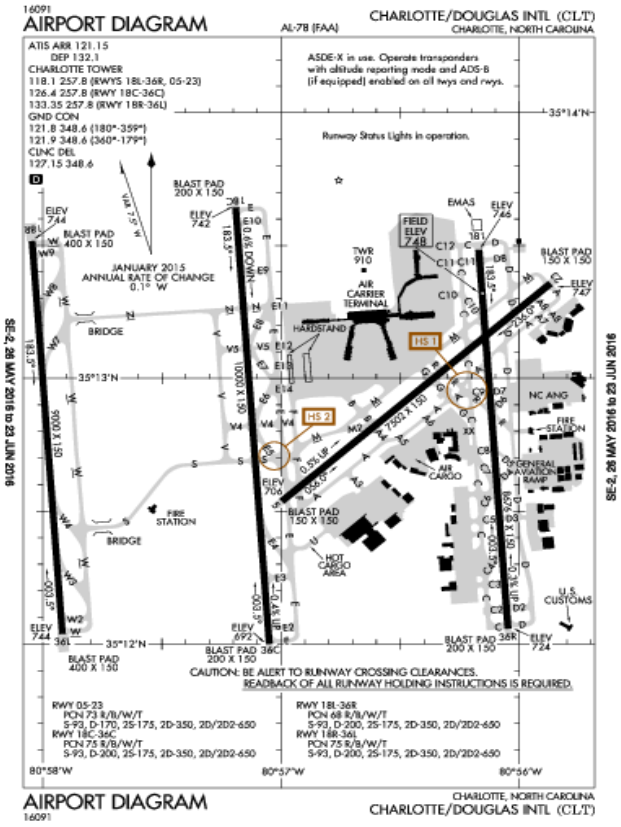

Figure 2: Charlotte Douglas International (CLT) Airport Diagram

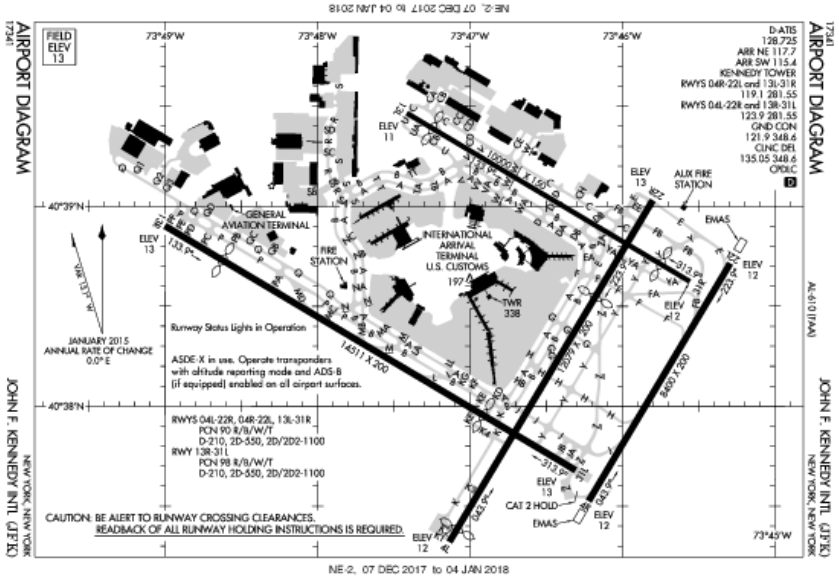

Figure 3: New York JFK International (JFK) Airport Diagram

For the ATP, many considerations are important in solving real problem. The first is the ability to model the system accurately relative to the real case. At any airport, each aircraft is assigned a gate during its stay at the airport. There is a scheduled departure time for each airplane to leave its gate. It is most preferable to leave the gate on schedule or even few minutes early, if all passengers have boarded. Of course, the flight might leave after its scheduled time too. Even when leaving the gate exactly on-time, there is a required time to taxi from the gate to the runway and possible waiting time in a queue of aircraft waiting for the runway to be cleared for takeoff. Having a model that captures these considerations is important to help the ATC to make the aircraft sequencing decisions. An additional task for ATC arises when there are multiple runways used for departure. In this case, runway assignment must be considered first.

Another important issue is to consider the basic standards required in the system such as the separation problem and definition of classes of aircrafts used by the ICAO. The separation requirement is very important between consecutive airplanes using the runway for safety reasons. When an airplane uses the runway to take off or land, a wake vortex is created from the movement of the aircraft and it can impose dangerous control issues for the next airplane using the runway. The effect of the wake vortex depends on both the size of the aircraft involved and the type of operation (landing or takeoff). Aircraft are classified either small, large or heavy. Large aircraft have larger wakes and thus require a longer separation time from the next aircraft. One of the main tasks for the ATC is to insure the required separation time between the aircraft using the runway for any given sequence. A standard minimum separation time between different classes of airplanes is enforced by both the FAA and the ICAO (summarized in Table 1).

One of the challenging issues regarding the ATP is defining the objective function, or the performance measure, for the problem. There are many stakeholders in this system, some with conflicting interests. An airline using the airport would prefer to have its flights leave on time with least cost possible (minimum waiting time to take off), the airport authority would prefer to have maximum throughput on its runways and achieve fairness among airlines using the airport. The FAA requires safe operation.

Table 1: Minimum Separation Times (in Seconds) for Two Consecutive Operations

a- Takeoff followed by takeoff

Leading/following	Heavy	Large	Small
Heavy	60	90	120
Large	60	60	90
Small	60	60	60

b- Landing followed by takeoff

Leading/following	Heavy	Large	Small
Heavy	40	40	40
Large	35	35	35
Small	30	30	30

c- Takeoff followed by landing

Leading/following	Heavy	Large	Small
Heavy	50	53	65
Large	50	53	65
Small	50	53	65

d- Landing followed by landing

Leading/following	Heavy	Large	Small
Heavy	99	133	196
Large	74	107	131
Small	74	80	98

Finally, it is important to have a method that solves the problem in the most efficient way and satisfies all of the stake holders. Also, it desirable to have the ability to solve the problem dynamically, as the uncertain and fluid nature of the problem based on the actual push back time, the taxi time, etc.

This dissertation is organized as follows. Chapter 2 is concerned with the literature and the approaches have been taken by previous researchers. Chapter 3 introduces practical approaches to solve the ATP problem for two types of aircraft using a single runway. Chapter 4 introduces a practical approach to solve the ATP problem for three types of aircraft using a single runway. Chapter 5 introduces a practical approach to solve the ATP problem for two types of aircraft using two parallel runways. Finally, Chapter 6 concludes with future work.

Chapter 2 - Literature Review

In this chapter, we review the literature on the runway scheduling problem. In Chapter 1, it was noted that the RSP, in general, is an NP-hard problem.

Although the main purpose is to maximize the efficiency of the system by improving the utilization of scarce resource, airport runways in this case, some studies consider different goals and different approaches to solve the problem. Different goals include minimizing the summation of delays or minimizing the summation of penalties for deviating from target landing time for the case of landing. Different models and solution approaches have been used to solve the RSP. In section 2.1, papers that deal with the RSP are categorized based on solution approaches with the emphasis on their objective functions and assumptions. In section 2.2, we introduce the machine scheduling literature relevant to the RSP problem, with the assumption of dependent setup times between different jobs, release times, and batching.

2.1 Runway Scheduling Problem Literature

The runway scheduling problem has been studied in the literature since the late 1970's. Psaraftis (1978) is one of the early studies in this area. Since then, different studies have considered different models, assumptions and goals. Also, different methodologies have been used to solve the RSP, for example dynamic programming, mixed integer programming, genetic algorithms, and many others. The review paper by Bennell et al. (2013) divided the area into two main categories based on the type of operation: aircraft landing problem (ALP), and aircraft take-off problem (ATP). However, since then, more papers consider both operations in the same model so it may be more appropriate to categorize by solution approach first.

Dynamic programming (DP) is one of the common approaches that have been used in the RSP. Because of the sequencing nature of the RSP problem, most cases can be modeled as a dynamic programming. In general, the time complexity for solving a single runway problem using DP is $O(CN^C)$, where N is the number of jobs and C is the number of aircraft classes (Bennell et al., 2013).

Bianco et al. (1999) solved the landing problem for one runway as a single machine scheduling problem. Sequence dependent setup times and release times were assumed in the

model to minimize completion time. A dynamic programming formulation was used to find a lower bound. Two heuristic algorithms were considered to solve the problem effectively for real scenarios.

Balakrishnan and Chandran (2007) presented a new class of techniques based on dynamic programming to optimize multiple objectives for a departure problem. They developed a unified framework under constraint position shifting (CPS). Other constraints such as wake vortex separation and runway crossing restrictions were considered.

Many other studies used dynamic programming to solve the problem. Psaraftis (1980), Trivizas (1998), and Craig et al. (2001) used dynamic programming to maximize throughput, while Lieder et al. (2015) used it to minimize delay. Many studies solved the ALP for multiple objectives using DP. Bayen et al. (2004) and Brentnall (2006) considered both throughput and delay for the ALP. Lee and Balakrishnan (2008), Balakrishnan and Chandran (2010), and Bennell et al. (2016) solved the ALP for multiple objective functions using DP.

Some studies considered local search heuristics for part or for the whole algorithm. Dear and Sherif (1989) is one of the early works that used a heuristic to maximize the runway throughput for the ALP, while maintaining constraint position shifting (CPS). Bianco et al. (2006) coordinates inbound/outbound traffic flows in the terminal and runway by using a job shop scheduling model with both setup time and release time constraints. They represent several operational constraints and runway configurations in a uniform framework. A fast-dynamic local search heuristic algorithm is used for the job shop model considering different performance measures. An experimental analysis for a real problem is considered.

Recently, Soomer and Frankx (2008), Furini et al. (2012, 2015), Sama et al. (2013, 2014, 2015), and Guepet et al. (2017) use heuristic methods to minimize the delay. Ma et al. (2014), Ghoniem et al. (2014), Ghoniem and Farhadi (2015), and Sabar and Kendall (2015) use heuristic methods in variant cases in the RSP for the same goal of minimizing the total deviation from target times. One of the issues with heuristics is that they do not guarantee optimality.

Another common approach in the RSP literature is genetic algorithms (GAs) and other metaheuristics. Atkin et al. (2007) used a hybrid meta-heuristic algorithm to recommend a schedule to aid decision making at the control tower. They mainly considered increasing

throughput for the take-off scheduling problem with the restrictions of separation times, controllers' limitations, and geometrical constraints. A case for London Heathrow airport is solved and compared to actual schedule performance.

Hancerliogullari et al. (2013) developed a priority rule for the mixed arrival and departure scheduling problem with multiple runways. The problem was modeled as an identical parallel machines scheduling model with release times, target and deadline times and sequence dependent constraints to minimize the weighted tardy times. A meta-heuristic (simulated annealing) was used to solve the problem and compared to optimal solutions for small and medium instances.

Other researchers used GAs to solve the ATP and the ALP. Some researchers use it in extended cases (e.g. multiple runways), as well. Beasley et al. (2001), Capri and Ignaccolo (2004), Veidal (2007), and Yu et al. (2011) used GAs to minimize the total deviation from target time for one runway landing cases. Stevens (1995), Ciesielsky and Scerri (1998), Cheng et al. (1999), Hansen (2004), Hu and Chen (2005), Pinol and Beasley (2006), Xie et al. (2013), and Zhou and Jiang (2015) employed GA for the ALP for two or more runways. While it is popular to use a GA in the RSP problem, the computational requirement make it difficult to find good solution in a reasonable amount of time.

Another common approach that have been used in the RSP is Mixed Integer Programing (MIP). It has been used to solve the problem since the 1990's. Beasley et al. (2000) considered the landing problem for both single and multiple runways to optimize multiple performance measures. A 0-1 MIP model using a tree search approach was used with flexible formulation and two possible objective functions: minimizing the cost of deviation from target time and maximizing throughput. A strong LP relaxation was developed to determine a lower bound. A heuristic algorithm was developed to improve the computation time. They solve up to 50 aircraft efficiently.

Anagnostacis et al. (2001) developed a framework for an automated decision aid to solve the ATP by increasing capacity (maximizing throughput) and decreasing delay, while Anagnostacis and Clarke (2003) considered maximizing runway utilization assuming the sequence of runway events (arrival, departure, and crossing) is given. They used a two-stage heuristic algorithm. In the first stage, the time for departure events is optimally allocated

given the time for arrival and crossing events. An IP model is used for the second stage to fill the departure events with the existing population of departing aircraft to maximize throughput.

Gupta et al. (2009) used an MIP model with multi-objective functions to minimize overall system delay, maximum individual delay, and maximize throughput for take-off. Both separation requirements and optional time window constraints were considered. They also considered different schemes for managing the queue area for the runway. A special case at Dallas Fort Worth (DFW) Airport was studied and a modified model (more computationally efficient) was presented.

Clare and Richards (2011) used an MIP model to optimize taxiway routing and runway scheduling throughput for the ATP. They reduced the average taxi time by half compared to first-come-first-served for a case at London Heathrow airport with up to 240 planes. An efficient method that reduces the computation time without loss in performance was introduced.

There are other studies considered using MIP or branch and bound methods to solve the RSP. Brinton (1992), Abela et al. (1993), Ernst et al. (1999), and Beasley et al. (2004) focused on minimizing total deviation from target times for landing aircraft. Wang et al. (2015) used branch and bound to minimize multi objective functions for landing, while Ghoniem et al. (2015), Vasilyev et al. (2016) and Avela et al. (2017) used an MIP to maximize runway throughput for mixed operation (landing and takeoff) cases. In general, MIP tends to have long computation times to reach an optimal solution. Also, as the number of aircraft becomes larger, the solution process grows exponentially.

Other methodologies used include ant colony optimization, queuing theory, column generation approach, simulation and constraint satisfaction. Van Leeuwen et al. (2002) used constraint satisfaction to assist controllers in planning and scheduling the ATP. Using the ILOG solver, an optimal departure schedule is found for maximizing throughput.

Randall (2002), Bencheikh et al. (2009), Feng et al. (2013, 2016), and Xu (2017) used ant colony optimization to minimize the deviation from target landing time, while Wen et al. (2005) used column generation for the same goal. In order to minimize delays, Pujet (1999)

and Bauerle et al. (2007) used queuing theory techniques, Brentnall and Cheng (2009) and Soykan (2016) used simulation, Murca and Muller (2015) used an exact algorithm approach, and Ma et al. (2016) used sparse optimization.

Some observations from the literature can be summarized as follow:

- RSP is an NP-hard problem. The presence of the separation time between different classes is one of the main reasons.
- Most used solution methods are: dynamic programming, genetic algorithm, mixed integer programming, and local search heuristics.
- Most common objectives: minimizing completion time and minimizing tardiness.
- Few studies achieve multiple optimization goals or have the flexibility to solve the problem for different goals.
- There is more literature about the ALP than the ATP. Most papers consider only one type of operation although some recent papers considered both cases in the same model.
- Most of the studies consider the static case while in reality, the RSP is dynamic. Since the RSP is a real case problem that can be observed in many large airports worldwide, it is desirable to have a solution that can be applied fast enough in the real cases while providing good quality answers. This can help the airport tower controllers to make better decisions.

Having these observations in mind, our goal is to solve the departure case runway scheduling problem efficiently for multiple objectives in a dynamic environment.

In the next section, the literature is discussed for the machine scheduling problem, especially those that share similarity with our model which is introduced in Chapter 3.

2.2 Single Machine Scheduling Literature

Because of the similarity between the machine scheduling and the runway scheduling problem, machine scheduling models can be used for the runway problem. The runway can be represented as a single machine while the aircraft can be represented as jobs to be processed on the machine.

There may be (literally) thousands of papers written on machine scheduling. Also, many review papers focusing on scheduling with specific constraints have been written. Allehverdi et al. (1999, 2008), and Allehverdi (2015) reviewed scheduling problems with setup considerations in different machine scheduling environments. Each paper divided the problems into independent setup and dependent setup in one dimension and batches and no batches in another dimension. Potts and Van Wassenhove (1992), Webster and Baker (1995), and Potts and Kovalyov (2000) reviewed scheduling with batching. They divided the problem into two main areas: family scheduling models and batching machine models.

Up to 2015, less than 80 papers considered having setup times and release times in various spectrums of problems from single machine to job shop, dependent or independent setups, and batch or non-batch format. Of those papers, few of them considered solving the problem for a single machine with dependent setup times and release times, as in our models in Chapter 3 and 4.

A summary table of the papers that considered dependent setup times and release times for a single machine environment is shown in Table 2. The solution method, and the objective function used in each paper is also shown in the table. A summary for each paper in the list is provided in Appendix B.

To our knowledge, there is no similar work in the literature using the same methodology as is used in this work to solve a similar problem.

Table 2: Summary for Single Machine Problems with Release and Dependent Setup Times Literature

Reference	Additional assumptions	Solution Approach	Objective
Farn and Muhlemann (1979)		Heuristics	∑Setup time
Bianco et al. (1988)		B & B	C_{max}
Raman et al. (1989)	Batches considered	Implicit Enumeration	∑F/n, ∑T/n
Uzsoy et al. (1992)	Precedence, DD (due date)	Heuristic	L_{max}, $\#U_j$
Ovacik and Uzsoy (1994a, b)	DD	Heuristic	L_{max}
Asano and Ohta (1996)	DD, no tardy jobs	B & B	∑Earliness
Asano and Ohta (1999)	DD, shutdowns	B & B	T_{max}
Sun et al. (1999)	Batches considered	Lagrangian relaxation (LR)	$\sum wT^2$
Baptiste (2000)	Batch availability, $p_j=p$	Dynamic programming	∑wU, ∑wC, ∑T, T_{max}
Shin et al. (2002)		Tabu search	L_{max}
Yuan et al. (2004)	Batch availability, $p_j=p$	Exact Algorithm	L_{max}
Baptiste and Le Pape (2005)	Batches considered, DD	B & B	∑f, f=cost of C
Chou et al. (2009)		Constraint programming, B&B, and two heuristics	∑wC
Nogueira et al. (2014)		MIP (6 formulations)	∑wC, ∑wT
Rojas-Santiago et al. (2014)		LR+2-opt Heuristic	C_{max}
Zhang and Xie (2015)	Group technology	Exact Algorithm for special cases	C_{max}

Chapter 3 – Practical Approaches for Solving the Single Runway Departure Sequencing Problem with Two Types of Aircraft

3.1 Introduction

In this chapter, the runway scheduling problem (RSP) is solved for take-offs (ATP) with one assigned runway and two types of aircraft. All flights are known and on time. The required time to travel from gate to runway is deterministic. Thus, the time when the aircraft is ready to take-off is known. A separation time between take-offs is required for safety. The required separation time varies between different types of aircraft as shown in Table 1 in Chapter 1. The objective function is to find an efficient schedule for two performance measures: minimizing completion time (C_{max}), and minimizing maximum delay (F_{max}). This facilitates providing all solutions on a dominant solution curve, allowing a decision maker (the tower controller) to implement schedules in near real-time.

These two objective functions are important measures in the RSP (de Neufville, 2016). Completion time is an appropriate measure to satisfy the runway capacity. In the other hand, maximum delay is an appropriate performance measure since fairness among different aircraft is desired. Note that an optimal solution for maximizing throughput may not lead to the optimal solution for minimizing maximum delay and vice versa.

There are a number of ways to find optimal solutions for the ATP. However, since the problem is NP-hard, Computational time grows exponentially with the size of the problem when using traditional optimization techniques (e.g. MIP and DP) as discussed in Chapter 2. The proposed solution method in this chapter provides optimal answers for both objective functions in a reasonable amount of time. There is also an opportunity to use the proposed algorithm to solve the problem for large cases, by dividing the problem into small sub-problems that can be solved separately.

In the next section, a scheduling model that represent the problem is introduced. Section 3.3 introduces a constructive approach to solve the problem for C_{max}. Section 3.4 describes a solution approach for minimizing F_{max} that solve the problem in reasonable time for a small case (up to 20 jobs), while section 3.5 is an extension to this approach to solve large cases. A computational study is performed in section 3.6.

3.2 Scheduling Model

Machine scheduling models can be used for the runway problem because of the similarity between the machine scheduling and the runway scheduling problem. The runway can be represented as a single machine while the aircraft can be represented as jobs to be processed on the machine. Before describing the problem via the scheduling model, some notation is needed to describe the model.

3.2.1 Notation

C_i Completion time of job i (take-off). $i = 1 \ldots N$.

C_{max} Largest completion time or completion time of last processed job $(C_{max} = max\{C_i\})$

C_{max}* Minimum C_{max} of all possible sequences

F_i Flow time of job i. In the RSP, it is the time between the release time r_i and take-off $(F_i = C_i - r_i)$.

$F_{average}$ Average flow time. $F_{average} = \sum F_i / N$

F_{max} Largest flow time. $F_{max} = max\{F_i\}$

F_{max}* Minimum F_{max} of all possible sequences

m_i Type of job i. In the RSP, it is the size of the aircraft (small, large, or heavy)

N Number of jobs

N_k Number of jobs from type k

p_i Processing time for job i. In the runway scheduling problem $p_i = p$ for all i.

r_i Release time of job i. The departure time from the gate plus the taxi time to the take-off point

S_{xy} Setup time between a job of type x followed by a job of type y. In the RSP, it is the required separation time between an aircraft of type x followed by an aircraft of type y.

t_i Starting time of processing job i. In the RSP, it is the time when the aircraft starts the process of taking-off.

Usage Total time of runway usage (i.e. idle time is excluded). $Usage = \sum p + \sum S_{xy}$.

3.2.2 Problem Description

Consider the following scheduling problem. There are N jobs. There is a single machine. Each job i has a time that it is released r_i to the production floor for processing. There are

two types of jobs, type 1, and type 2. If a type 2 job precedes a type 1 job, there is a setup time S_{21} added after the processing of the type 2 job. If a type 1 job precedes a type 2 job, there is no setup time. There is no setup time between jobs of the same type. All processing times are known and constant.

The first objective is to identify a schedule that minimizes C_{max}. The second objective is to identify a schedule that minimizes $F_{max.}$ While minimizing C_{max} is desirable, it may be appropriate to identify schedules with larger C_{max}, to minimize F_{max}, as treating each job (aircraft) fairly is also important. Thus, minimizing F_{max} could be considered an alternative (secondary) objective of the problem. As a consequence, considering multiple schedules is important.

Clearly, if one ignores the release times r_i, the optimal solution is to group jobs by type and then schedule small aircraft followed by large ones (as that minimizes C_{max}). However, considering the release times, this approach may lead to idle time in the schedule.

When solving for C_{max}, there may be multiple optimal solutions for C_{max}. In this case, a better candidate would be the one with less runway usage. *Usage* can be defined as total processing time plus total setup (separation) time (i.e. idle time is excluded). Having lower runway usage means more idle time, thus, more room for additional flights in the future.

Following the scheduling formulation $\alpha/\beta/\gamma$ as described in (Pinedo 2008) where α represents the machine environment, β represents the processing characteristics and constraints, and γ represents the objective to be minimized, the problem can be described as:

$1/\{r_i, S_{21}, p_i = p\}/\{C_{max}, F_{max}\}$, where *1* refers to the single machine environment.

Table 3 Describes the problem as a scheduling model.

Table 3: Problem description as a scheduling model

Real Problem	Scheduling Model
• One runway	• Single machine (N jobs)
• Two types of aircraft	• Two job types (m_1, m_2)
• Ready times (when aircraft ready to take-off) are known.	• Each job i has release time r_i
• Sequence dependent separation time between take-offs.	• Processing times p are known and constant • If a type 2 precedes a type 1, a setup time S_{21} is added after the type 2 job • Otherwise no setup time
• Two objectives: Maximize throughput (C_{max}), Minimize maximum delay (F_{max}).	• $C_{max} = max\{C_i\}$, C_i: Completion time of job i • $F_{max} = max\{F_i\}$, $(F_i = C_i - r_i)$

3.3 A Constructive Method for Solving the ATP to Minimize Completion Time (C_{max})

In order to solve the described scheduling problem in sec 3.2 for minimizing C_{max}, a constructive algorithm is used. Basic rules to build the algorithm are introduced first, followed by algorithm description.

3.3.1 Build Up for the Method

Let $\sum p$ = summation of process time for all jobs in a sequence, $\sum S$ = summation of all separation times required between jobs, and $\sum d$ = summation of all idle times occurs between jobs. Separation time occurs when switching from one type to the other according to the definition of the problem. Idle time occurs when the machine is idle and next job has not yet been released.

For $1/\{r_i, S_{21}, p_i = p\} / C_{max}$, $C_{max} = \sum p + \sum S + \sum d$. Since $\sum p$ is fixed, then minimizing C_{max} is equivalent to minimizing $\sum S + \sum d$. Since $p=1$, $S= \{0 \text{ or } 1\}$ and release times are integers, the following rules minimize $\sum S + \sum d$:

1. Ordering each type jobs by their release time will minimize idle time in the problem.
2. If there are two job types available while the machine is idle, starting with the one that needs less separation time will always reduce the number of setups in the problem.

3. Sequencing jobs of the same type, if no idle time is left between them, will minimize the setup time for the problem.

3.3.2 Minimizing C_{max} Algorithm

Using the rules described above in the scheduling problem described in section 3.2.2, the algorithm can be described as follows:

1. Start from $t = r_1$ (where r_1 is the release time of first available job in the problem)
2. Consider a list for all jobs available at time t, ordered by their release times:
    a) If there is only one type of jobs available (Type 1 or 2), start with it.
    b) If more than one type is available, start with a type 1 job.
3. Take the sequenced job out of the list.
4. Move to $t = C_{[j]}$ (where $C_{[j]}$ is completion time of last job in the sequence)
5. Update the list by considering all available jobs at time $t = C_{[j]}$
    a) If the first available job in the list after completion of *job [j]* is of the same type of job *j*, then, sequence this job next.
    b) If the first available job is not of the same type, then, if the first available job from the same type has a release time less than or equal $C_{[j]}$ (for type 1) or $C_{[j]} + S_{21}$ (for type 2) sequence it after job *j*. Otherwise, sequence the first available job from the other type.
    c) If no jobs (from both types) are available, then move to $t = r_k$, where *k* is the first available job. Repeat step 2 for $t = r_k$.
6. Repeat steps 2 to 5 until all jobs sequenced.

To illustrate how the algorithm works, consider the following example:

$N = 10$, $p = 2$, $S_{21} = 1$, $S_{12} = 0$, release times and types of jobs are in Table 4.

Table 4: Data for the Example for Minimizing C_{max} Algorithm

Job #	1	2	3	4	5	6	7	8	9	10
Release time	0	0	2	4	6	7	8	10	11	13
Type	1	2	1	2	1	1	2	2	1	2

At time $t = 0$, there are two available jobs: one of type 1, and one of type 2. In this case, the type 1 job will be sequenced first. Next, at time $t = 2$ (completion time of first sequenced job), there are two available jobs (2 and 3). Since the first available job (2) is not of the same type as the sequenced job, we consider job 3. Since r_3 is less than or equals C_1, job 3 is sequenced. Next, at time $t = 4$, there are two jobs of type 2 but no available type 1 jobs. In this case, job 2 is sequenced after job 3. Repeating this process until all jobs sequenced will lead to the sequence in Figure 4 with $C_{max} = 21$.

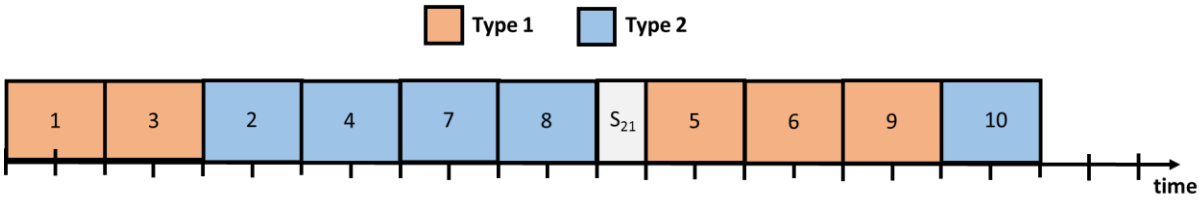

Figure 4: Final Sequence for the C_{max} Algorithm Example

Note that this approach does not consider minimizing F_{max} in the first place. This means that it is possible to obtain an optimal sequence for C_{max} but not optimal for F_{max}. Minimizing F_{max} is considered in the next section.

3.4 An Exact Approach for Solving the ATP to Minimizing Maximum Delay (F_{max})

In order to solve the described scheduling problem in sec 3.2 to minimize F_{max}, an exact algorithm is considered. The main idea of the algorithm is based on using the specific characteristics of the ATP to identify a relatively small number of feasible solutions from all possible solutions of the problem. From there, it is guaranteed that the optimal solution for minimizing F_{max} is within the identified candidate solutions.

For any scheduling problem, a solution is a sequence of jobs. One simple (but not necessarily practical) way to find the optimal sequence is to enumerate all sequences and find the optimal one. The number of possible sequences in this case is $(N!)$, where N is the number of jobs.

In Section 3.4.1 the necessary principles needed to build the algorithm are introduced. Section 3.4.2 describes how the candidate optimal solutions are identified, while section

3.4.3 explains the stopping criteria. Section 3.4.4 describes the algorithm, section 3.4.5 provides an example, and section 3.4.6 identifies the efficiency of the approach

3.4.1 Basic Principles to Build the Algorithm

For $1/ r / C_{max}$, ordering by release date (ORD) is optimal (Pinedo 2008). Since $p_j = p$ is a special case, then ORD is also optimal for $1/ \{r, p_j=p\} / C_{max}$.

F_{max} can be considered as a special case of L_{max}, with the due date for each job equals its release time. It is known that ORD is not guaranteed optimal for $1/r/L_{max}$ (Pinedo 2008). However, for the case of $1/ r / F_{max}$, ORD is optimal. It is also optimal for $1/ \{r, p_j=p\} / F_{max}$

Theorem 3.1: ORD is optimal for $1/ \{r, p_j=p\} /F_{max}$.

Proof: Consider a schedule S where all jobs are ordered by release time. Considering only two adjacent jobs i and j in this schedule $(p_i = p_j = p \text{ and } r_i < r_j)$ where job i starts at time t,

$F_{max}(i, j) = max [C_i–r_i, C_j–r_j]$, noting that $C_i = t +p$ and $C_j = t +2p +k$ (where $k = r_j-C_i$, *if* $r_j > C_i$, 0 otherwise), then $F_{max}(i, j) = max [t +p –r_i, t+2p+k–r_j]$.

Now, consider a schedule S' where all jobs are ordered by release time except job i and j, which have been reversed. Considering only jobs i and j in this schedule $(p_i = p_j = p \text{ and } r_i < r_j)$ where job j starts before job i, then,

$F_{max}'(i, j) = max [C_j'–r_j, C_i'–r_i]$, noting that $C_j'= t +p +m$, and $C_i'= t+2p+m$ (where $m = r_j–t$ if $r_j > t$, 0 otherwise), then $F_{max}'(i, j) = max [t +p +m –r_j, t +2p +m –r_i]$

Now there are three possible scenarios:

1- If $r_j > C_i$, then, r_j is also greater than t $(C_i > t)$, $k > 0$, $m > 0$ and $m > k$.

In this case, $t +2p +m –r_i$ is greater than both $t +p –r_i$, and $t +2p +k –r_j$.

2- If $C_i > r_j > t$, then, $k = 0$ and $m > 0$.

In this case, $t +2p +m –r_i$ is greater than both $t +p –r_i$, and $t +2p –r_j$.

3- If $r_j < t$ then, $k = 0$ and $m = 0$.

In this case, $t + 2p –r_i$ is greater than both $t +p –r_i$, and $t +2p –r_j$.

In all three cases, $F_{max}'(i, j) > F_{max}(i, j)$. This means switching job *i* and *j* will never improve F_{max} for these two jobs. In addition, switching any two adjacent jobs may increase the completion time for other jobs (as in case 1 and 2 above), which may make F_{max} worse.

Thus, ordering all jobs by release time will optimize F_{max} for any schedule. Q.E.D.

Going back to the runway problem, it means that if only one type of aircraft is considered, then ordering them by release time will provide the optimal schedule for both C_{max} and F_{max}. Notice that if 2 or more jobs have the same release time, it is optimal if they are ordered arbitrarily.

Now consider the problem $1/\{r_i, S_{21}, p_i = p\}/\{C_{max}, F_{max}\}$ (described in section 3.2.2)

Theorem 3.2: for $1/\{r_i, S_{21}, p_i = p\}/\{C_{max}, F_{max}\}$, If all jobs from type 1 must be processed first, followed by all jobs from type 2, or vice versa, then ordering each type of jobs by their release time is optimal for C_{max} and F_{max}.

Proof: Consider the case where all type 2 jobs are processed first, followed by all type 1 jobs (note there is a setup time S_{21} between the two types).

Since it is not allowed to sequence any type 1 jobs before the type 2 jobs are sequenced, then sequencing type 1 jobs is independent from sequencing type 2 jobs. Thus, ORD is optimal for the first group (containing type 2 jobs), similar to $1/r/C_{max}$. Let the completion time for this schedule called C_{max}^1.

For the second group (type 1 jobs), jobs cannot be processed until the completion of the last type 2 job plus the setup time (S_{21}). There are two cases. First, if all the release times of the type 1 jobs (2nd group) are greater than $C_{max}^1 + S_{21}$, then the two parts can be considered separate problems. Therefore, ORD must be optimal for this group. In the other case, if more than one job from type 1 has a release time smaller than C_{max}^1, then the two groups are not separate. However, since it is restricted to start any job from the second group before the completion of the first group, an adjusted release time (r_j') for the jobs that have a release time smaller than C_{max}^1 can be defined as: $r_j' = C_{max}^1 + S_{21}$. In this case, this part becomes equivalent to the $1/r/C_{max,}$ with possible multiple jobs having similar release times at the beginning. Thus, ORD is optimal (ties are broken arbitrarily) with a completion time C_{max}^2.

Finally, consider the completion time for the whole problem, C_{max}. Since both parts of the problem can be optimal by ORD, it is clear that C_{max} for the whole problem is optimal too.

Note that a similar argument can be used when starting with type 1 jobs followed by type 2 jobs (except setup time is zero). The same argument can be applied for the case of F_{max}. Q.E.D.

In general, when there are M types of jobs, $M = \{g_1, g_2, \ldots, g_m\}$, and all other assumptions are the same, the following theorem can be stated.

Theorem 3.3: For $1/\{r, S_{gh}, p_j = p\}/\{C_{max}, F_{max}\}$, where S_{gh} = setup time when type g job followed by type h job, if each type's jobs must be processed together, then ordering each group by its release times is optimal for C_{max} and F_{max} regardless of the groups order.

Proof: Let $M = \{g_1, g_2 \ldots, g_m\}$ be the number of types. Without loss of generality, assume all jobs of type g_1 are processed first, followed by jobs of type g_2, and so on. From Theorem 3.2, ORD is optimal for first group g_1. It is also optimal for the 2nd group. All remaining groups have similar situation to the 2nd group, therefore, ORD in each group is optimal for C_{max}. The same argument applies for F_{max}. Q.E.D.

The next step is to develop a rule when one of the types (say type 1) is partitioned into two separate groups. In this case, the machine will process the first group of the jobs of type 1, then, it switches to process all the available jobs of type 2, then, go back to the first type and process the rest of the jobs.

Partitioning one of the families (types) into two separate groups will add two new decisions: 1) What is the size of each group? and 2) Which group should each job be assigned? For the second decision, a simple answer would be to assign the jobs based on their release dates (for example if the first group contains k jobs, assign k jobs with the smallest release time to this group). However, since the first and third group contains jobs from the same type, swapping jobs between them will not add setup times in addition to those already exist in the sequence, yet, it is not clear if swapping any jobs can improve the solution (i.e. assigning first k jobs might not be optimal). A rule can be established in this case.

In order to establish a rule, consider the problem $1/r/C_{max}$. Assume the machine, besides processing jobs of type 1, will process one other job of, say type 2. This job has a release

and a processing time. A setup time *(S_{21})* is needed when switching from processing jobs of type 2 to jobs of type 1. Also, this job must be processed after processing *k* jobs of type 1. This means that type 1 jobs will be partitioned into two separate groups with *k* jobs in the first group and the remaining in the latter group. In this case, ORD for type 1 jobs is optimal for C_{max}, by assigning the first *k* jobs, ordered by their release times, to the first group and the remaining jobs, ordered by their release times, to the latter group. The same holds for F_{max}.

Theorem 3.4: Let $\{r_1^1, r_2^1, \dots, r_k^1, r_{k+1}^1, \dots, r_m^1\}$ be the release times of *m* jobs of type 1 such that $r_1^1 \leq r_2^1 \dots \leq r_m^1$. Also, let r_1^2 be the release time of a type 2 job.

For *1/ {r, S_{21}, $p_j = p$} / {C_{max}, F_{max}}.* If the type 2 job must be processed after a specified number (*k*) of jobs from type 1, then sequencing jobs with release times: $r_1^1, r_2^1, \dots, r_k^1$,ordered by their release time, before the type 2 job and jobs with release times: $r_{k+1}^1, \dots, r_m^1$, ordered by their release time, after the type 2 job is optimal for C_{max} and F_{max}.

Proof: Assume that the jobs with release times $r_1^1, r_2^1, \dots, r_k^1$ are assigned to the first group and jobs with release times $r_{k+1}^1, \dots, r_m^1$ are assigned to the latter group. Since all the jobs in the first group must be processed before the jobs in the second group, it becomes equivalent to the case described in theorem 3.3. Therefore, ordering jobs in each group by their release time is optimal for C_{max}.

From this result, swapping any job other than the k^{th} from the first group with any job in the other group will not improve the performance (it will violate the ORD rule inside one or both groups). Similarly, swapping any job other than the $k+1^{th}$ from the second group with any job from the first group will not improve the performance. However, swapping job *k* from the first group with job *k+1* from the second group will not violate the ORD rule for each group (or swapping jobs *{k-1, k}* with *{k+1, k+2}* and so on)

Notice that when swapping job *k* with job *k+1*, any setup times S_{12} and S_{21} will remain the same. Notice also that all processing times are equal. This indicates that there will be no effect on the performance due to the setup times and process times. Now, since $r_{k+1}^1 \geq r_k^1$, swapping job *k+1* with job *k* will never decrease the completion time of the first group (depending on the position of r_k^1 and r_{k+1}^1 on the timeline, the completion time might increase or remain the same). Similarly, the completion time of the latter group will never

decrease when swapping job *k+1* with *k*. Thus, the completion time for the whole sequence (C_{max}) will never decrease. The same argument holds if swapping two or more jobs (e.g. swapping jobs *{k+1, k+2}* with *{k-1, k}*).

This indicates that swapping jobs *k* and *k+1* will never improve the performance of C_{max}, which also indicates that ordering by release time will guarantee optimality. The same argument applies for F_{max}. Q.E.D.

Note that if $p_i \neq p$, ordering type 1 jobs by release times, when a type 2 job must be processed after *k* jobs of type 1, is not necessary optimal. Consider the following counter example (3 type-1 jobs and a type-2 job with release times and processing times as shown in Table 5. Type-2 jobs must be processed after 2 type-1 jobs. $S_{12} = 1$, $S_{21} = 0$).

Table 5: Data for the Counter Example

Job j	Type	r_j	p_j
1	1	0	2
2	1	1	1
3	1	2	2
4	2	5	2

If the type 1 jobs are ordered by release times, as shown in Figure 5(a), then, $C_{max1} = 9$ units. However, a better solution with $C_{max2} = 8$ units is possible with the sequence *{1,3,2}* as shown in Figure 5(b).

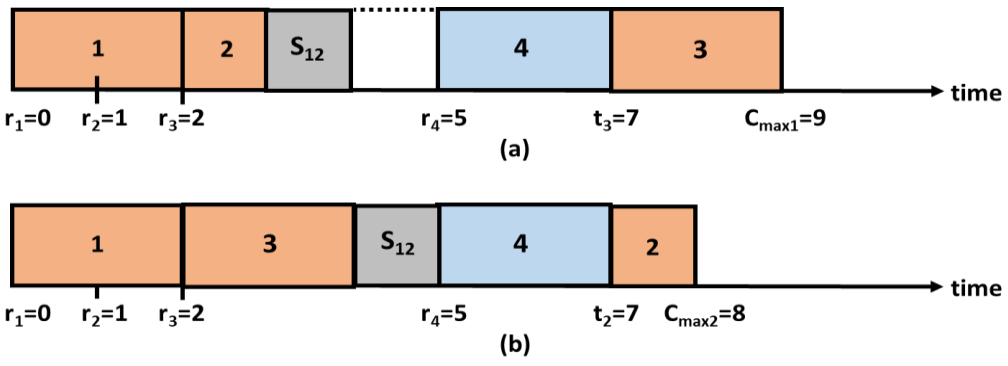

Figure 5: Gantt Chart for the Counter Example

In the RSP, since the processing times are equal, the result obtained from Theorem 3.4 is important for constructing the proposed exact algorithm. If the processing times are not equal, DP can be used to solve the problem (Wang and Uzsoy, 2002).

In general, if more than one job of type 2 is in a middle group that partitions the family of type 1 into two separate groups, then, the same rule above should apply as explained in the next theorem.

Theorem 3.5: Let $\{r_1^1, r_2^1, \dots, r_k^1, r_{k+1}^1, \dots, r_m^1\}$ be the release times of *m* jobs of type 1, and $\{r_1^2, r_2^2, \dots, r_n^2\}$ be the release times of *n* jobs of type 2 such that $r_1^1 \leq r_2^1 \dots \leq r_m^1$ and $r_1^2 \leq r_2^2 \dots \leq r_n^2$.

For *1/ {r, S_{21}, p_j = p} / {C_{max}, F_{max}}*, If a specified number of jobs (*k*) from type 1 must be processed first, followed by all jobs of type 2, followed by the remaining jobs of type 1, then:

1) assigning jobs with release times $r_1^1, r_2^1, \dots, r_k^1$ to the first group and jobs with release times $r_{k+1}^1, \dots, r_m^1$ to the third group, and

2) ordering each group of jobs by their release time, is optimal for C_{max} and F_{max}.

Proof: Assume that the jobs with release times $r_1^1, r_2^1, \dots, r_k^1$ are assigned to the first group, ordered by their release times, and jobs with release times $r_{k+1}^1, \dots, r_m^1$ are assigned to the third group, ordered by their release times, while the jobs of type 2 are assigned to second group, ordered by their release times. In this case, from theorem 3.3, ORD in each group will minimize the completion time of each group. Also, from theorem 3.4, swapping any jobs from the first group and the third group will not improve the results if only one type 2 job is in the second group. The same argument holds if more than one type 2 jobs are sequenced in the middle group (the completion time might increase or remain the same but never decrease, in any of the groups). Thus, the current sequence is optimal for C_{max}. The same argument holds for $F_{max.}$ Q.E.D.

Finally, Theorem 3.6 can be generalized for *G* partitions as stated in the next theorem.

Theorem 3.6: Let $\{r_1^1, r_2^1, \dots, r_m^1\}$ be the release times of *m* jobs of type 1, and $\{r_1^2, r_2^2, \dots, r_n^2\}$ be the release times of *n* jobs of type 2, such that $r_1^1 \leq r_2^1 \dots \leq r_m^1$ and $r_1^2 \leq r_2^2 \dots \leq r_n^2$.

For $1/\{r, S_{21}, p_j = p\}/\{C_{max}, F_{max}\}$, If the sequence is partitioned into G groups between the two types, such that (m_1) jobs from type 1 must be processed in the first group, followed by (n_1) jobs of type 2 in the second group, followed by (m_2) jobs of type 1 in the third group and so on, until the last group (G) with all the jobs assigned, then:

1) assigning jobs to each group according to their release time order, and

2) ordering each group of jobs by their release time, is optimal for C_{max} and F_{max}.

Proof: Assume all jobs are assigned to groups based on their release time order. From Theorem 3.4, ORD is optimal for C_{max} in each group. From Theorem 3.5, swapping any jobs between groups containing the same type of jobs will never improve the performance of C_{max}. Therefore, generalizing the case for G groups can hold for optimizing C_{max}. Same argument holds for F_{max}. Q.E.D.

This means if the number of jobs in each group is specified, then, ORD in each group is optimal for C_{max} and F_{max}. However, if the number is not specified then there is no simple rule to solve the problem. This is the next step in the algorithm.

In general, when a specific sequence of groups is considered and each group contains a specific number of jobs of one type, then, the setup times in the problem is controlled. This means, any changes in the sequence will not eliminate setups.

Also, when the process times are all equal, then, regardless of any idle time occurrences (because of the release times), the performance cannot be improved by swapping any jobs between groups. So, instead of focusing on the release times or the setup time, we quickly identify candidate solutions with a simple rule, and then evaluate each one by their performance.

Note that if more than two types of jobs exist, the number of possible partitions will be much larger as the problem grows. An alternative approach is to consider dynamic programming (Wang and Uzsoy, 2002). However, because only two types of jobs are considered, it is easy to identify all possible partitions of the scheduling problem. Identifying them is done in two stages: 1) identify the number of groups/partitions, and 2) identify all possible number of jobs in each group/partition.

3.4.2 Identifying the Number of Possible Solutions

In order to describe the process of identifying the number of possible solutions the following definition is introduced.

Definition 3.1: A **sequence family** contains all the sequences that have the same number of groups formed by the two types of jobs in the problem. For example, a sequence family for 3 groups contains all the sequences that may be formed by starting with the first type followed by the second type followed by the first type (or start with the second type, followed by first type, followed by second type) with all the possible number of jobs in each group.

The identification of possible solutions is done in two stages: 1) identify all the possible sequence families, and 2) identify all candidate sequences that fit in each sequence family.

1) Identifying all possible sequence families:

Suppose there are N jobs (m_1 of these jobs are type 1 while $m_2 = N-m_1$ are type 2). A *group* of jobs is defined as a number of jobs of the same type that are sequenced consecutively, and with no job of the same type sequenced immediately before or after the group. Note that a group can consist of one job if a job of one type is preceded and followed by jobs of another type. Thus, for any sequence of jobs, there is only one possible way to identify the groups in the problem. For example, a schedule with the following job types sequence *{1, 1, 1, 2, 2, 1, 2, 2, 1, 1}* has five unique groups.

From these outcomes, it is possible to define a sequence family from each group formation. In order to find the number of all possible sequence families, there are two cases: (1) if the number of jobs is equal for both types or (2) if the number of jobs is not equal. For both cases, the minimum number of groups is two (one for each type of jobs). The maximum number of groups when there is an equal number of jobs of each type is N (where N is the total number of jobs and each group consists of one job only). Thus, the total number of different sequence families is $2(N-1)$.

For the non-equal case, all sequence families from 2 to $2m$ groups are possible formations (where m is the number of jobs from the type which has the least jobs). Since each sequence family has two different scenarios (starting with type 1 or type 2), there will be at least $2(2m - 1)$ possible formations. An additional formation can be considered when starting and

ending with the jobs of the type with greater number of jobs, while having m groups of the other type, each consists of 1 job. This makes the possible number of sequence families =

[2(2m – 1)] +1 = 4m – 2+1 = 4m – 1.

Identifying the number of sequence families is useful to identify the total number of sequences in each family to be examined, which is discussed next.

2) Identifying the number of candidate sequences within each sequence family

When there are two types of jobs, the number of possible solutions for each sequence family can be identified. There are two cases to consider: (1) when the number of jobs in each type is equal and (2) when the number of jobs is not equal.

For the first case,

The number of solutions = $2*\left[\sum_{i=1}^{\frac{N}{2}}\binom{\frac{N}{2}-1}{i-1}\binom{\frac{N}{2}-1}{i-1}+\sum_{i=1}^{\frac{N}{2}-1}\binom{\frac{N}{2}-1}{i-1}\binom{\frac{N}{2}-1}{i}\right]$, $N > 2$

For the second case, let N_1= number of jobs from type 1, N_2 = number of jobs from type 2

$N_1' = \max\{N_1, N_2\}$, $N_2' = \min\{N_1, N_2\}$

The number of solutions =

$$2*\sum_{i=1}^{N_2'}\binom{N_1'-1}{i-1}\binom{N_2'-1}{i-1}+\sum_{i=1}^{N_2'}\binom{N_1'-1}{i}\binom{N_2'-1}{i-1}+\sum_{i=1}^{N_2'-1}\binom{N_1'-1}{i-1}\binom{N_2'-1}{i}, N_2' > 1$$

Table 6 compares the number of possible solutions using the group search algorithm versus total enumeration, which is *N!*

Table 6: Computational Reduction for the Search Algorithm

No of jobs	Search algorithm	Total enumeration
8	70	40,320
12	924	4.79 E+8
16	12,870	2.09 E+13
20	184,756	2.43 E+18

3.4.3 Identifying a Stopping Criteria

When the search is ordered by number of groups, the value of F_{max} follows a convex function as the number of groups increases. The reason for this is because as the number of groups

increases, the amount of setup time increases. At the same time, the amount of idle time may decrease. The result is a convex curve. This result can be used to identify a stopping point in the search as following: When the optimal value for F_{max} for k groups is less than optimal value for F_{max} for $k+1$ groups, it means that F_{max} for k groups is optimal for the problem. Thus, the search can be stopped at this point.

3.4.4 Describing the Algorithms

The solution procedure is described as follows:

1- Determine the number of all sequence families for this problem (as described in 3.4.2)
2- Start with the sequence family of two groups starting with first type job followed by the second type.
3- Using Theorem 3.6 to identify candidate sequences, consider all the sequences for jobs that can fit within this sequence family (Note that for two-group sequence family, starting with first type of job, only one sequence is possible. That is, to start with all jobs from the first type ordered by their ready time followed by the jobs from the second type ordered by their ready time. However, for more than two groups, use the formula described in 3.4.2 - part 2 to count the number of all candidates)
4- Evaluate all candidate schedules in this sequence family and pick the one with best solution as a candidate solution for this group formation.
5- If it is feasible, reverse the order of types while staying in the same sequence family and repeat steps 3 to 4. Otherwise go to step 7.
6- For each sequence family pair, with the same number of groups, consider the best solution of the two candidates.
7- Consider the next sequence family (by increasing the number of groups by one). Repeat steps from 3 to 6.
8- Continue searching each sequence family pair until the candidate solution for the current pair is larger than the candidate solution for the previous pair. Label the candidate solution for the previous pair as S_0 and the candidate solution for the current pair as S_1.

9- If the candidate solution for the next pair (say S_2) is also larger than S_0 then STOP. The optimal solution for the problem is S_0. Otherwise go to step 7 until all sequence families are searched.

3.4.5 Applying the Algorithm in a Numerical Example

MATLAB was used to implement the algorithm. See Appendix E for details of the code.

In order to apply the algorithm, the following example is introduced.

p = 2 units. S_{21} = 1 unit. S_{12} = 0 unit.

Table 7 shows the release time and type of each job.

Table 7: Input Data for F_{max} Algorithm Example

Job, i	1	2	3	4	5	6	7	8	9	10
Release time, r_i	0	1	2	4	5	6	8	10	11	12
Type, m_i	2	2	1	1	2	2	1	1	2	1

If ordering by release time or first come first served (FCFS) is used, then, C_{max} = 23, F_{max} = 11, and the total number of groups = 6.

Using the proposed algorithm, the candidate optimal solution for each group formation for the example is shown in Table 8, while Figure 6 represents the candidate solutions in an XY plot.

Table 8: Candidate Optimal Solutions

No. of groups	C_{max}* Search		F_{max}* Search	
	Candidate C_{max}	Associate F_{max}	Candidate F_{max}	Associate C_{max}
2	24	14	14	24
3	**21**	**13**	11	22
4	**22**	**10**	10	22
5	22	11	11	22
6	**23**	**11**	11	23
7	23	12	12	23
8	**24**	**12**	12	24
9	24	13	13	24
10	**25**	**13**	13	25

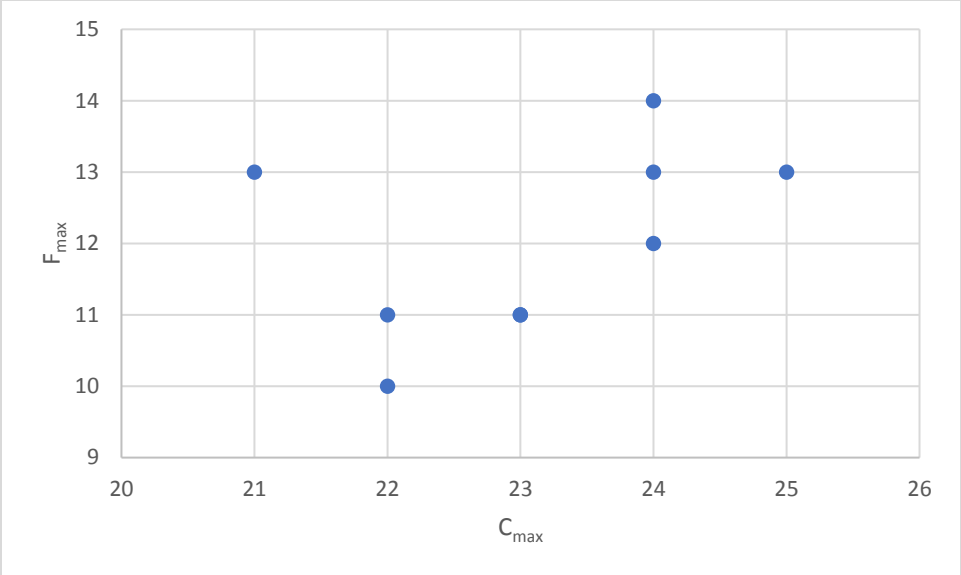

Figure 6: Candidate Optimal Solutions in XY Plot

From Table 8, if the algorithm is used to find C_{max}*, then, C_{max}* = 21, F_{max} = 13, and the total number of groups = 3. *S1* = {1, 2, 3, 4, 7, 8, 10, 5, 6, 9}.*

In the other hand, if the algorithm is used to find F_{max}*, then, C_{max} = 22, F_{max}* = 10, and the total number of groups = 4. *S2* = {1, 3, 4, 2, 5, 6, 9, 7, 8, 10}*

A summary for all three solutions is in Table 9. Figure 7 illustrates the candidate schedules for (a) FCFS. (b) C_{max}* sequence. (c) F_{max}* sequence.

Table 9: Example Summary Outcome

	No of Groups	C_{max}	F_{max}
FCFS	6	23	11
C_{max}* search	3	21	13
F_{max}* search	4	22	10

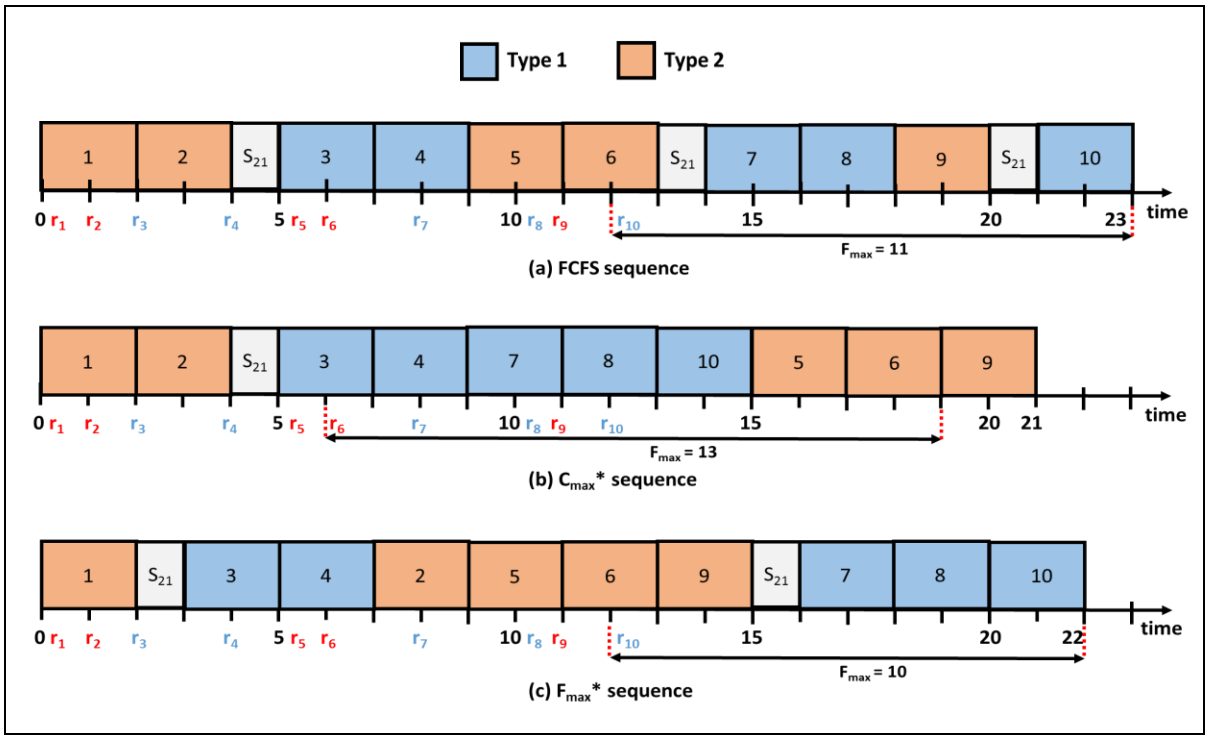

Figure 7: Gantt Chart for the Example

The first insight from this example is that FCFS does not guarantee a good solution. Although it might look like a reasonable choice, the time added switching from one group type to the other, which is accumulated by adding more separation time, might be worse than the idle time eliminated from ordering by release times.

It is also important to notice that the optimal sequence for minimizing C_{max} is different than the optimal one for minimizing F_{max}. For this reason, it is important to consider one of the measures as the primary objective function while keeping the other as a secondary measure. Another option is to consider one of the measures as a constraint limit for the problem while optimizing the other. In the RSP, it is reasonable to minimize the completion time while protecting the fairness among the aircraft by not allowing any aircraft to be delayed for more than certain limit.

3.4.6 Efficiency of the Approach

In order to measure the efficiency of this approach, an experiment was performed in MATLAB. Randomly generated cases were created considering different number of jobs. In

each case, 10 runs (with random release times and aircraft types) for a specific number of jobs was measured in term of CPU time. Table 10 summarizes the results. Since it is desired to solve the problem in quickly (a few seconds), a problem of up to 18 jobs can be reasonably solved using this approach.

Table 10: Computation Time Versus Number of Jobs

No of jobs	10	12	14	16	18	20	22	24
Average CPU (Sec)	0.23	0.33	0.66	1.35	4.23	16.39	43.30	224.47

3.5 An Extension to the Exact Approach to Solve the ATP for Minimizing Maximum Delay (F_{max}) for Large Cases

The proposed algorithm in section 3.4 solves the problem for a small number of jobs. However, it is necessary (in the aircraft take off application) to solve the problem for large numbers of jobs, and for dynamic cases, where some information may not be available immediately. In this section, the algorithm in section 3.4 is modified to handle such cases. This is done by dividing the problem into small sets. Each small set of the problem can be solved separately until the whole problem is solved.

The approach solves the planned schedule for all given jobs. However, changes happen (some jobs will have different actual release times than scheduled release times). Yet this approach can handle changes, by considering a new problem with the new changes and solving the new problem in the same approach (without rescheduling the already scheduled jobs)

Section 3.5.1 provides a key observation for solving this problem and section 3.5.2 gives the modified algorithm to solve the problem for large cases.

3.5.1 Analyzing the Behavior of the Algorithm When Increasing Number of Jobs

In order to the algorithm for the dynamic case, it is desirable to understand the behavior of the ATP when increasing one job at a time. Consider the following example:

For a problem with: $N = 20$ jobs, $S_{12} = 4$, $S_{21} = 2$, $p_j = 1$ for all j, the release times, r_j, and job types, m_j, are given in Table 11.

Table 11: Data for the Example

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
r_j	0	1	2	8	10	12	14	17	21	22	28	32	38	41	44	45	47	51	53	59
m_j	2	2	1	2	2	1	2	1	2	1	1	2	1	2	1	2	1	1	2	2

Table 12 shows the optimal solution for F_{max} for different instances starting from the first three jobs up to 20 jobs. In each instance, the optimal sequence, the optimal number of groups, the value of F_{max}* and the job having F_{max} are presented in the table. Note that type 1 jobs are in parenthesis.

An important observation from this experiment is that the sequence stabilizes as the number of jobs increases. This indicates that it might be possible to divide the problem into sub-problems. Each sub-problem can be solved separately while maintaining optimality for all active aircraft. This result contributes to the solution approach in two aspects: (1) computation times can be drastically reduced (asymptotic linear time), and (2) the dynamic environment of the ATP can be handled in real time.

Table 12: Output Analysis

# jobs	Optimal Sequence	F_{max}*	# groups
3	**1 2 (3)**	**3**	2
4	**1 2 (3) 4**	**3**	3
5	**1 2 (3) 4 5**	**3**	3
6	**1 2 (3) 4 5 (6)**	**3**	4
7	**1 2 (3) 4 5 (6) 7**	**5**	5
8	**1 2 (3) 4 5 (6) 7 (8)**	**5**	6
9	**1 2 (3) 4 5 7 (6 8) 9**	**6**	5
10	**1 2 (3) 4 5 7 (6 8) 9 (10)**	**6**	6
11	**1 2 (3) 4 5 7 (6 8) 9 (10 11)**	**6**	6
12	**1 2 (3) 4 5 7 (6 8) 9 (10 11) 12**	**6**	7
13	**1 2 (3) 4 5 7 (6 8) 9 (10 11) 12 (13)**	**6**	8
14	**1 2 (3) 4 5 7 (6 8) 9 (10 11) 12 (13) 14**	**6**	9
15	**1 2 (3) 4 5 7 (6 8) 9 (10 11) 12 (13) 14 (15)**	**6**	10
16	**1 2 (3) 4 5 7 (6 8) 9 (10 11) 12 (13) 14 16 (15)**	**6**	10
17	**1 2 (3) 4 5 7 (6 8) 9 (10 11) 12 (13) 14 16 (15 17)**	**6**	10
18	**1 2 (3) 4 5 7 (6 8) 9 (10 11) 12 (13) 14 16 (15 17 18)**	**6**	10
19	**1 2 (3) 4 5 7 (6 8) 9 (10 11) 12 (13) 14 16 (15 17 18) 19**	**6**	11
20	**1 2 (3) 4 5 7 (6 8) 9 (10 11) 12 (13) 14 16 (15 17 18) 19 20**	**6**	11

In order to modify the algorithm to solve the dynamic case (and become faster for the static case) it is important to identify the following questions:

1- How many jobs to consider in each set?
2- Is there a rule to determine the jobs that should be sequenced in the current schedule (to facilitate dynamic scheduling)?

Answering these questions will help to identify rules that can be used to build an efficient algorithm for the dynamic case.

3.5.2 Describing the Extension Algorithm

The proposed algorithm solves the problem by starting with sequencing small set of jobs. Then, new jobs are added to the set and resolved until all jobs are sequenced. The number of jobs considered in the set each time is fixed. A different number of sets is considered in the experiment to measure quality of approach.

Using the same scheduling model notations described in section 3.2.1 and problem definition in section 3.2.2, the new algorithm can be described as following:

1- Order all jobs by release time. Consider the first *K* jobs
2- Solve them using small size algorithm (section 3.4). pick the best sequence (with best F_{max} and best $F_{average}$ as a tie breaker)
3- Commit only jobs that have completion time less than release time of first job not considered yet *r(K+1)*:
    a. If all jobs in the current set are completed before *r(K+1)*, then all jobs in the current set are fixed and next *K* jobs will be considered for the next set.
    b. If *J* out of *K* jobs are completed before *r(K+1)*, then only these jobs are fixed and the remaining *(K-J)* jobs are carried-on to the next set
    c. If no jobs are completed before *r(K+1)*, then only the first completed job will be fixed to insure continuity.
4- Consider the initial conditions for next set.
5- Repeat 2 to 4 until all jobs sequenced

3.6 Computational Study

To test the proposed methods against the typical (FCFS) method for scheduling departing aircrafts of two types using one runway, a computational experiment using New York JFK airport data is performed. The data considered from New York - JFK airport at peak hours (From 5pm to 7:15pm). 100 aircrafts scheduled to depart. There are two types of aircrafts: large (type 1) and heavy (type 2). Single runway is considered for departure. We assume all departures on time and taxi time is not considered. If a type 2 is followed by type 1 in the sequence, a separation time of 1 minute is required between them Otherwise, no separation time is needed. Processing time for each job is 1 minute.

To examine the new approach in problems with smaller release times span, there are two considerations for the release times in this case:

1) Actual release times
2) Modified release time where span of release times is reduced by 20% (i.e. $r_j' = 0.8^* (r_j - r_1) + r_1$, rounded to nearest minute)

A different number of sets is considered in the extended approach to measure its quality. The details of actual and modified data (schedule times and types) are provided in appendix C.

The performance measures are: completion time *(C_{max})*, runway utilization *(Usage)*, maximum delay (F_{max}) and average delay ($F_{average}$). CPU time is considered to measure procedures quality. Three approaches considered for the problem:

1) First come first served (FCFS).
2) The constructive method (C_{max}* Approach) from section 3.3 focusing on minimizing C_{max}.
3) The extended method (F_{max}* Approach) from section 3.5 focusing on minimizing F_{max}.

The experiments were conducted using MATLAB. The summary of results is in Table 13 (All measures in minutes except *CPU* time in seconds). In Table 13, the columns are defined as following:

(C_{max}) = completion time of all aircrafts

(*Usage*) = total (processing times + separation times) (i.e. idle time excluded)

(*U%*) = *Usage* / C_{max}

(F_{max}) = Maximum flow time for an aircraft

(F_{avg}) = Average flow time of all aircrafts

(*CPU*) = Computation time (in seconds)

In both cases (Actual release time and modified release time), using C_{max}* approach and F_{max}* approach provide better results for F_{max} and *Usage* compared to FCFS approach. Also, for F_{max}* approach, as the number of jobs considered in each set increases, the performance of F_{max} and F_{avg} improves. However, CPU time increases as set size increases.

Comparing C_{max}* approach with F_{max}* approach, the first approach reduced the *Usage* to 110 compared to 112 in the second approach for the actual release time case. However, F_{max} was better using the second approach (11 compared to 13). Similarly, for the second case (with modified release times), *Usage* was reduced to 106 using C_{max}* approach compared to 108 using F_{max}* approach, while F_{max} was better using the second approach (11 compared to 15).

Table 13: Experiment Summary

Proc.	Actual release times						Modified release times (reduced by 20%)					
	C_{max}	*Usage*	*U%*	F_{max}	F_{avg}	*CPU*	C_{max}	*Usage*	*U%*	F_{max}	F_{avg}	*CPU*
FCFS	139	123	88.49	13	5.37	0.02	123	123	100	20	11.13	0.02
C_{max}*	137	110	80.29	13	4.02	0.12	112	106	94.64	15	5.13	0.14
F_{max}* Set=2	137	115	83.94	11	4.40	0.21	117	117	100	14	7.14	0.17
F_{max}* Set=4	137	113	82.48	11	4.20	0.51	114	114	100	13	6.27	0.34
F_{max}* Set=6	136	112	82.35	11	4.08	0.67	113	113	100	12	5.99	0.49
F_{max}* Set=8	136	112	82.35	11	4.06	0.79	113	112	99.12	12	5.89	0.71
F_{max}* Set=10	136	112	82.35	11	4.06	0.94	112	108	96.43	11	5.28	1.03
F_{max}* Set=12	136	112	82.35	11	4.06	1.07	112	108	96.43	11	5.28	1.24
F_{max}* Set=14	136	112	82.35	11	4.06	1.95	112	108	96.43	11	5.28	1.76

3.7 Conclusion

The goal in this chapter is to provide an approach that can solve the described problem effectively with flexibility. Effectiveness can be achieved by solving the problem in few seconds for a large case (such as 100 jobs) with efficient results for multiple performance measures under the given input. Flexibility is achieved by providing two approaches with different objective for each. This provide the decision maker an opportunity to choose the desired objective as needed

As shown in the computational experiments, both algorithms were effective. Both algorithms can solve the problem practically in few seconds to provide better output compared to FCFS for a real case such as JFK airport. Each approach provided better output for one of the performance measures (C_{max} or F_{max}) compared to FCFS. In addition, both approaches perform better under tighter reduced release times compared to FCFS.

In conclusion, using the real problem characteristics to consider the proposed solution approaches was a key factor. Having two aircraft types with specific separation times made it possible to introduce a cut technique and quick search approach. Also, spread of release times of jobs made it possible to solve separate sets.

Chapter 4 – A Practical Approach for Solving the Single Runway Departure Sequencing Problem with Three Types of Aircraft

4.1 Introduction

While some major airports in the US deal with two types of aircraft (large and small) for most of the day, they may operate three types during other times, especially in the evening when many heavy aircraft (e.g. Boeing 777) travel to overseas destinations. With this motivation, in this chapter scheduling three types of A/C on a single runway is studied.

When three types of A/C are considered, the number of possible ways to divide the sequence into groups based on the A/C types is large. However, if number of A/C from one of the types is relatively small compared to the other two types, it may be reasonable to modify proposed algorithm presented in Chapter 3 to solve for three types of A/C.

Section 4.2 describes the problem using a scheduling model. Section 4.3 introduces an approach to solve the problem for C_{max} or F_{max} for small instances and a modified approach to solve large cases in a reasonable amount of time. A computational study is performed in section 4.4 and the conclusion is given in section 4.5.

4.2 Scheduling Model and Problem Description

Similar to Chapter 3, a machine scheduling model can be used for the single runway problem. The runway can be represented as a single machine while the aircraft can be represented as jobs to be processed on the machine. In this case, three types of jobs are considered instead of two. Using the same notation as in section 3.2.1, the problem can be described as follows.

There are N jobs. There is a single machine. Each job i has a time that it is released, r_i, for processing. There are three types of jobs, type 0, type 1 and type 2. There is a setup time S_{xy} added after the processing of a type x job and its value depends on the type of the following job. The complete values for S_{xy} are shown in Table 14. All processing times p are known and constant. The objective is to identify schedules that minimize C_{max} and F_{max}.

Table 14: FAA Separation Requirements (in Minutes)

Leading/ following	Heavy	Large	Small
Heavy (type 2)	0	1	2
Large (type 1)	0	0	1
Small (type 0)	0	0	0

Following the scheduling formulation, $\alpha/\beta/\gamma$ as described in (Pinedo 2008), where α represents the machine environment, β represents the processing characteristics and constraints, and γ represents the objective to be minimized, the problem can be described as:

$1/\{r_i, S_{21}, S_{20}, S_{10}, p_i = p\}/\{C_{max}, F_{max}\}$, where 1 refers to the single machine environment.

4.3 Solution Approach for Three Types Problem

In order to minimize C_{max} or F_{max} for the scheduling problem in section 4.2 an exact algorithm is considered. The main idea of the algorithm is based on using the specific characteristics of the ATP to define the problem as a sequence of groups instead of sequence of jobs. Each group, contains a number of aircraft (jobs) of one type. From there, it is shown that for each group formation, there is one guaranteed optimal solution for this group formation. Thus, it is guaranteed that if all group formations are identified, the optimal solution for minimizing C_{max} or F_{max} is within the identified solutions.

While it was possible to identify a stopping point in the search for the two job types problem in Chapter 3, it is more complicated with three job types available, and multiple possible values for S_{xy}.

4.3.1 Building Blocks for the Proposed Algorithm

Consider the problem described in section 4.2: $1/\{r_i, S_{21}, S_{20}, S_{10}, p_i = p\}/\{C_{max}, F_{max}\}$. Let $\{r_1^0, r_2^0, \dots, r_m^0\}$ be the release times of m jobs of type 0, $\{r_1^1, r_2^1, \dots, r_n^1\}$ be the release times of n jobs of type 1, and $\{r_1^2, r_2^2, \dots, r_q^2\}$ be the release times of q jobs of type 2, such that $r_1^0 \leq r_2^0 \dots \leq r_m^0$, $r_1^1 \leq r_2^1 \dots \leq r_n^1$ and $r_1^2 \leq r_2^2 \dots \leq r_q^2$.

Theorem 4.1: For $1/\{r_i, S_{21}, S_{20}, S_{10}, p_i = p\}/\{C_{max}, F_{max}\}$, if the sequence is partitioned into G groups between the three types, such that jobs from type 0 must be processed in X groups (m_1 jobs in group X_1, m_2 jobs in group X_2, and so on, where $m = m_1 + m_2 + \dots$ and $X = X_1 + X_2 + \dots$), similarly, jobs of type 1 must be processed in Y groups, and jobs of type 2 must be

processed in Z groups, such that total number of groups (G) = $X + Y + Z$ and no consecutive groups contains same type of jobs, then for a given sequence of the groups:

1) assigning jobs to each group in release time order, and

2) ordering jobs within a group by release time, is optimal for C_{max} and F_{max}.

Proof: A similar theorem was proven in Chapter 3 for the two job types case (Theorem 3.6). Since swapping any jobs between groups of the same type will not improve C_{max}, regardless of the type of jobs in the groups in-between as stated in Theorem 3.4 then release time dispatching also holds for three job types. The same argument holds for F_{max}. QED.

This means if the number and order of groups are specified and if the number of jobs in each group is also specified, then, ORD in each group is optimal for C_{max} and F_{max}.

For the three A/C type problem, identifying all possible partitions formation for the scheduling problem (by specifying the number and order of groups and number of jobs in each group) can be counted as follows:

Let X = number of type 0 jobs, Y= number of type 1 jobs, and Z = number of type 2 jobs.

$N = X + Y + Z$, is the total number of jobs

Then, the number of possible group formations = $\binom{N}{X} * \binom{N-X}{Y}$

Table 15 and Figure 8 show the number of possible solutions for N = 12 with different numbers of type 0 (X), and type 1 (Y). (Note that number of type 2 jobs (Z) = N-X-Y.) Table 16 shows the number of possible solutions for various instances with N =16.

Table 15: Number of Possible Solutions for N = 12

No of type 0 jobs (X)	No. of type 1 jobs (Y)				
	1	2	3	4	5
1	132	660	1980	3960	5544
2	660	2970	7920	13860	16632
3	1980	7920	18480	27720	27720
4	3960	13860	27720	34650	27720
5	5544	16632	27720	27720	16632

Table 16: Number of Possible Solutions for N = 16

# Type 0 jobs (X)	# Type 1 jobs (Y)						
	1	2	3	4	5	6	7
1	240	1,680	7,280	21,840	48,048	80,080	102,960
2	1,680	10,920	43,680	120,120	240,240	360,360	411,840
3	7,280	43,680	160,160	400,400	720,720	960,960	960,960
4	21,840	120,120	400,400	900,900	1,441,440	1,681,680	1,441,440
5	48,048	240,240	720,720	1,441,440	2,018,016	2,018,016	1,441,440
6	80,080	360,360	960,960	1,681,680	2,018,016	1,681,680	960,960
7	102,960	411,840	960,960	1,441,440	1,441,440	960,960	411,840

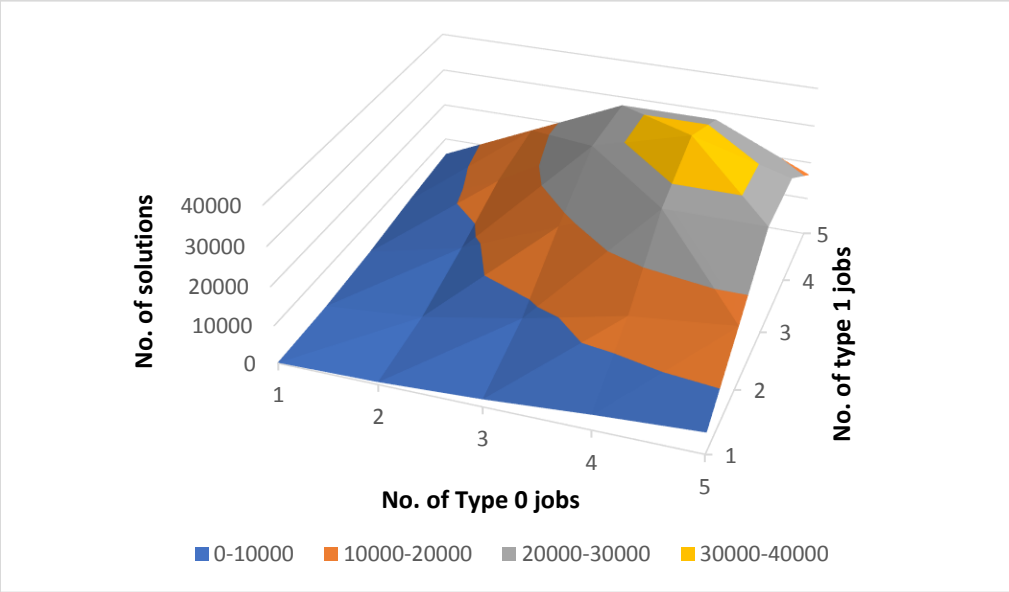

Figure 8: Number of Possible Solutions for N = 12

4.3.2 A Proposed Algorithm for Small Instances

The algorithm for small instances can be described as follows:

1- Identify all candidate solutions: To generate all possible candidates, consider the following: X = number of type 0 jobs, Y= number of type 1 jobs, and Z = number of type 2 jobs. $N = X + Y + Z$

Then, number of solutions = $\binom{N}{X} * \binom{N-X}{Y}$

2- Form all possible candidates. A possible candidate is where each type's jobs are ordered by release time (but not necessary that all jobs are ordered by release time).

3- Evaluate each candidate solution and pick the optimal one. In case of ties, the average flow time of all aircraft (F_{avg}) is used as a tie breaker for $F_{max,}$ while the runway utilization *(Usage)* is used as a tie breaker for C_{max}.

As shown in Table 15 and 16, the number of possible searches increases exponentially as the number of jobs increase (for most (*X, Y*) combinations). Thus, using this algorithm is practical for small instances (solvable in few seconds for up to 14 jobs). An extension of the algorithm to solve large instances is introduced next.

4.3.3 Solution Approach for Large Instances

Similar to the case of two A/C types, it is possible to divide the problem into small sets that can be solved separately. The proposed algorithm for three types solves the problem by starting with sequencing a small set of jobs. Then, new jobs plus carried-on jobs from previous set are considered in new set and solved until all jobs are sequenced. The number of jobs considered in the set is fixed. The modified algorithm can be described as following:

1- Order all jobs by release time. Consider the first *K* jobs as the first set.
2- Solve the set using the small instances algorithm (Section 4.3.2).
3- Commit only jobs that have completion time less than release time of first job not considered yet *r(K+1)*:
    a. If all jobs in the current set are completed before *r(K+1),* then all jobs in the current set are fixed and next *K* jobs will be considered for the next set.
    b. If *J* out of *K* jobs are completed before *r(K+1),* then only these jobs are fixed and the remaining *(K-J)* jobs are carried-on to the next set
    c. If no jobs are completed before *r(K+1),* then only the first completed job will be fixed to insure continuity.
4- Move to the next set.
5- Repeat steps 2, 3 and 4 until all jobs are sequenced.

4.4 Computational Study

To test the proposed methods against the typical (FCFS) method for scheduling departing aircrafts of three types using one runway, a computational experiment using New York JFK airport data is performed. The data considered from New York - JFK airport is at peak hours

(from 5pm to 7:15pm). In the data there are 100 aircraft scheduled to depart. There are three types of aircraft: small (type 0), large (type 1) and heavy (type 2). A single runway is used for departure.

We assume all departures to be on time and taxi time is not considered. If a type 2 followed by type 1 in the sequence, a separation time of 1 minute is required between them. If a type 1 followed by type 0, a separation time of 1 minute is required. If a type 2 followed by type 0, a separation of 2 minutes is required. Otherwise, no separation time is needed. Processing time for each job is 1 minute. Table 14 shows the separation times between different types.

Two different cases are considered to examine the performance of solution approaches:

1) Actual data (the ratio for type 0 to type 1 to type 2 aircrafts is approx. 1:5:4).
2) Similar number of jobs and schedule times to actual data but a ratio of approximately 2:4:4 for A/C types.

For each case, two considerations for the release times:

1) Actual release times
2) Modified release time where the span of release times is reduced by 20% (i.e. $r_j' = 0.8^* (r_j - r_1) + r_1$, rounded to nearest minute)

The details of actual and modified data (schedule times and types) are provided in appendix C. The performance measures are runway utilization (*Usage*), maximum delay (F_{max}) and average delay (F_{avg}). CPU time is used to measure procedure quality.

There are three approaches considered:

1) First come first served (FCFS),
2) F_{max}* approach: the extended method focusing on minimizing F_{max}, and
3) C_{max}* approach: the extended method focusing on minimizing runway usage.

Different set sizes are considered in the second and third objectives to measure quality of each approach. The experiments were conducted using MATLAB (see appendix F for codes). The summary of results is in Tables 17 and 18 (All measures are in minutes except CPU time, which is in seconds). The columns in both tables are defined as follows:

(C_{max}) = completion time of all aircraft

(*Usage*) = total (processing times + separation times) (i.e. idle time excluded)

(U%) = Usage/C_{max}

(F_{avg}) = Average flow time of all aircraft (tie breaker for C_{max}* and F_{max}*)

(F_{max}) = Maximum flow time for an aircraft

(CPU) = Computation time, in seconds

In the first case (with types ratio of 1:5:4 with actual release times from the data), the runway usage improved from 132 minutes (FCFS), to 119 minutes when minimizing $F_{max,}$ and to 118 minutes when minimizing C_{max}, the maximum delay (F_{max}) improved from 14 minutes (FCFS) to 11 minutes when minimizing $F_{max,}$ and to 13 minutes when minimizing $C_{max.}$

Table 17: Experiment Summary - Case 1: Actual Data with 1:5:4 Types Ratio

Approach	Actual release times						Modified release times (reduced by 20%)					
	C_{max}	*Usage*	*U%*	F_{max}	F_{avg}	*CPU*	C_{max}	*Usage*	*U%*	F_{max}	F_{avg}	*CPU*
FCFS	139	132	94.96	14	6.54	0.02	132	132	100	28	15.57	0.02
F_{max}* Set size =2	137	123	89.78	13	4.86	0.18	123	123	100	20	11.18	0.14
F_{max}* Set size =4	137	121	88.32	12	4.41	0.33	120	120	100	17	9.72	0.37
F_{max}* Set size =6	136	120	88.24	12	4.34	0.33	117	117	100	16	8.64	0.55
F_{max}* Set size =8	136	120	88.24	12	4.30	0.65	115	115	100	15	7.69	1.09
F_{max}* Set size =10	136	119	87.50	11	4.20	1.35	113	113	100	12	6.20	3.20
F_{max}* Set size =12	136	119	87.50	11	4.20	5.73	113	113	100	12	6.07	7.95
C_{max}* Set size =2	137	122	89.05	17	4.71	0.15	119	119	100	18	9.55	0.19
C_{max}* Set size =4	136	119	87.50	17	4.27	0.21	113	113	100	19	7.15	0.24
C_{max}* Set size =6	136	118	86.76	16	4.28	0.25	112	112	100	18	6.16	0.40
C_{max}* Set size =8	136	118	86.76	16	4.29	0.62	112	112	100	20	5.90	0.85
C_{max}* Set size =10	136	118	86.76	13	4.13	1.28	112	111	99.12	20	5.58	2.37
C_{max}* Set size =12	136	118	86.76	13	4.22	5.15	112	110	98.21	18	5.46	8.94
C_{max}* Set size =14	136	118	86.76	13	4.16	33.44	112	110	98.21	28	5.56	81.36

Table 18: Experiment Summary - Case 2: Actual Data with 2:4:4 Types Ratio

Approach	Actual release times						Modified release times (reduced by 20%)					
	C_{max}	*Usage*	*U%*	F_{max}	F_{avg}	CPU	C_{max}	*Usage*	*U%*	F_{max}	F_{avg}	CPU
FCFS	140	138	98.57	16	7.36	0.03	138	138	100	34	18.98	0.02
F_{max}* Set size =2	138	129	93.48	13	5.01	0.20	127	127	100	23	12.84	0.14
F_{max}* Set size =4	138	126	91.30	13	4.84	0.25	124	124	100	20	11.52	0.44
F_{max}* Set size =6	137	125	91.24	12	4.61	0.34	119	119	100	16	9.07	0.59
F_{max}* Set size =8	137	123	89.78	11	4.37	0.71	117	117	100	15	7.90	1.41
F_{max}* Set size =10	137	123	89.78	11	4.36	3.56	116	116	100	14	7.38	5.94
F_{max}* Set size =12	137	123	89.78	11	4.36	15.61	116	116	100	14	7.38	22.43
F_{max}* Set size =14	137	123	89.78	11	4.36	121.95	115	115	100	13	7.00	139.62
C_{max}* Set size =2	138	126	91.30	13	4.86	0.15	122	122	100	19	10.44	0.15
C_{max}* Set size =4	137	121	88.32	17	4.63	0.24	114	114	100	19	7.78	0.27
C_{max}* Set size =6	137	120	87.59	16	4.49	0.32	114	114	100	17	6.70	0.39
C_{max}* Set size =8	137	120	87.59	13	4.31	0.73	113	112	99.12	18	6.30	1.02
C_{max}* Set size =10	137	119	86.86	13	4.25	2.97	113	112	99.12	19	6.21	4.88
C_{max}* Set size =12	137	119	86.86	13	4.31	14.45	113	112	99.12	19	6.18	20.07
C_{max}* Set size =14	137	119	86.86	13	4.27	113.05	113	111	98.23	25	6.19	127.65

In the first case (release times reduced by 20%), the runway usage improved from 132 minutes (FCFS) to 113 minutes when minimizing F_{max}, and to 110 minutes when minimizing C_{max}, the maximum delay (F_{max}) improved from 28 minutes (FCFS) to 12 minutes when minimizing F_{max}, and to 18 minutes when minimizing $C_{max.}$

In the first case (with types ratio of 1:5:4 with actual release times from the data), the runway usage improved from 132 minutes (FCFS), to 119 minutes when minimizing $F_{max,}$ and to 118 minutes when minimizing C_{max}, the maximum delay (F_{max}) improved from 14 minutes (FCFS) to 11 minutes when minimizing $F_{max,}$ and to 13 minutes when minimizing $C_{max.}$

In the first case (release times reduced by 20%), the runway usage improved from 132 minutes (FCFS) to 113 minutes when minimizing F_{max}, and to 110 minutes when minimizing

C_{max}, the maximum delay (F_{max}) improved from 28 minutes (FCFS) to 12 minutes when minimizing F_{max}, and to 18 minutes when minimizing $C_{max.}$

In the second case (with types ratio of 2:4:4 with actual release times from the data), the runway usage improved from 138 minutes (FCFS) to 123 minutes when minimizing F_{max}, and to 119 minutes when minimizing C_{max}. The maximum delay (F_{max}) improved from 16 minutes (FCFS) to 11 minutes when minimizing F_{max} and to 13 minutes when minimizing $C_{max.}$

In the second case (release times reduced by 20%), the runway usage improved from 138 minutes (FCFS) to 115 minutes when minimizing F_{max}, and to 111 minutes when minimizing C_{max}. The maximum delay (F_{max}) improved from 34 minutes (FCFS) to 13 minutes when minimizing F_{max}, and to 25 minutes when minimizing C_{max}. Figure 9 and 10 summarize these results for each case.

In each case, using C_{max}* and F_{max}* approaches provide better results for F_{max} and usage compared to FCFS. However, C_{max}* approach outperforms F_{max}* approach in improving runway usage, while F_{max}* approach outperforms C_{max}* approach in improving maximum delay.

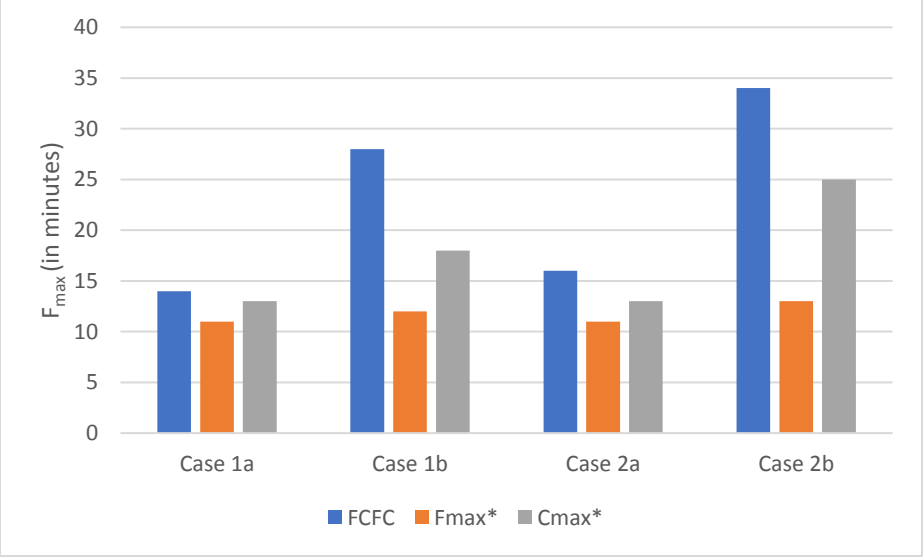

Figure 9: Comparing F_{max} Performance Using FCFS, F_{max}* and C_{max}* Approaches

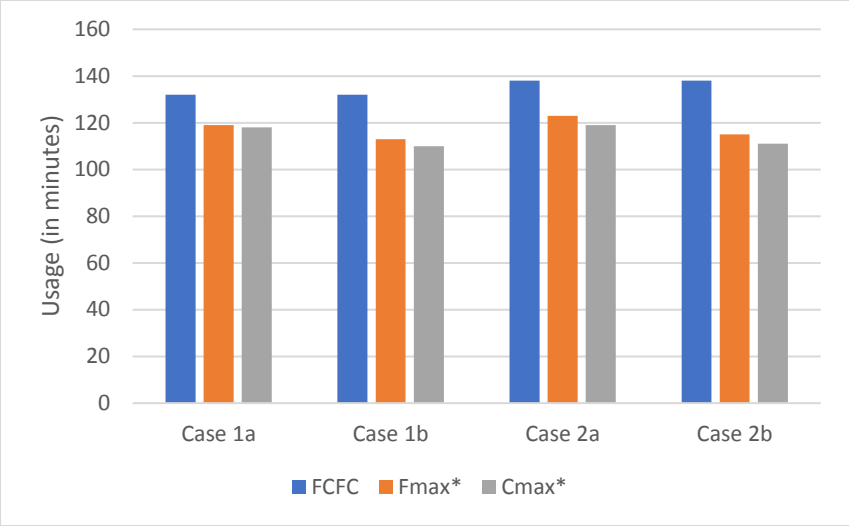

Figure 10: Comparing *Usage* Performance Using FCFS, F_{max}* and C_{max}* Approaches

For the C_{max}* and F_{max}* approaches, as the number of jobs considered in each set increases, the performance of usage and F_{max} improves, respectively. However, CPU time also increases as set size increases. Figure 11 shows the relation between set size and values of C_{max}, F_{max}, and CPU time, when using F_{max}* approach for the second part of the second case (2:4:4 types ratio with reduced release times). Thus, the approach is limited to practically solve instances of up to 14 jobs. However, using the approach to solve sets for up to 14 jobs is reasonable to solve the ATP problem as results from experiment has shown.

When considering 2:4:4 types ratio and condensed release times (as in the second part of the second case), it is possible to have a dominant solutions curve between *Fmax* and *usage* (Figure 12 shows the dominant solutions for case 2b). This can provide the decision makers (control tower) an efficient tool to maximizing one of the measures or balance the output between both as desired. If it is more important to prioritize throughput over maximum delay, then, solving the problem using *Cmax** approach will be the desired choice, and vice versa.

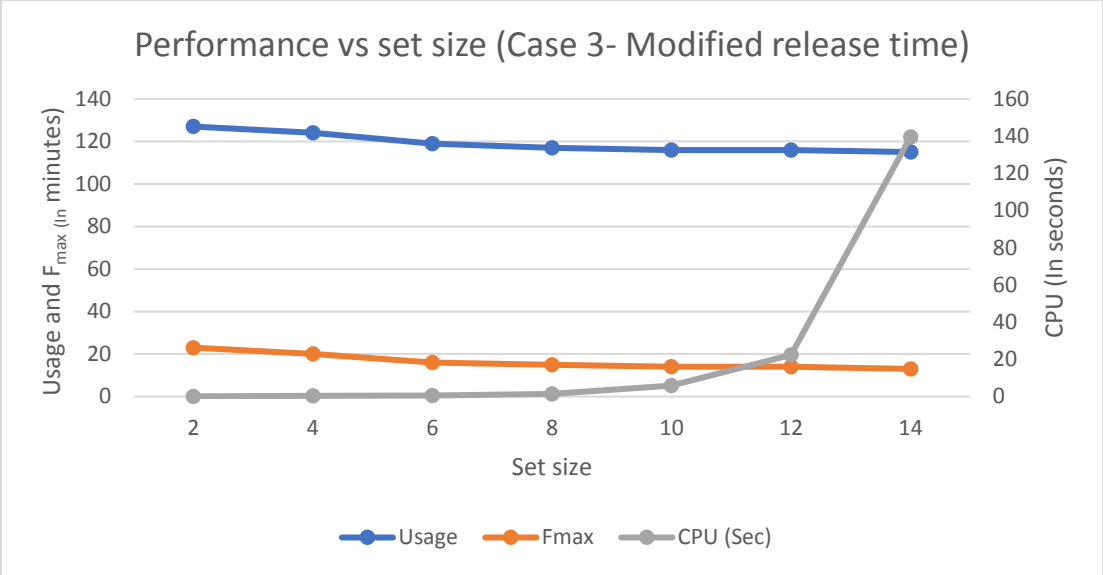

Figure 11: Performance vs Set Size for Case 2b (Types Ratio of 2:4:4 and Modified Release Time)

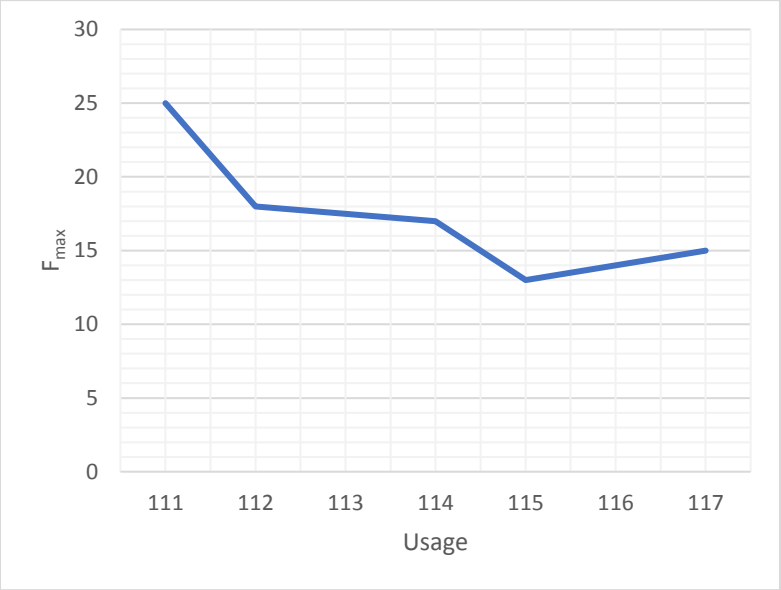

Figure 12: Dominant Solutions for *Usage* and *Fmax* (Case 2b)

4.5 Conclusion

The goal in this chapter is to provide an approach that can solve the problem of scheduling 3 types of departing aircraft on one runway effectively and with flexibility. Effectiveness is achieved by solving the problem in just a few seconds for a large real case (100 jobs) with

efficient results for multiple performance measures. Flexibility is achieved by providing two approaches for multiple performance measures, C_{max}* and F_{max}*.

As shown in the computational experiments, both algorithms were effective. Both algorithms can solve the problem in a few seconds to provide better performance compared to FCFS for a real case (JFK airport). Each approach provided better output for one of the performance measures (usage or F_{max}) compared to FCFS. In addition, both approaches perform better under heavier runway loads and diverse A/C types ratios, compared to FCFS.

In conclusion, using the ATP problem characteristics as the proposed solution approach may be a key contribution to developing more efficient airport operations.

Chapter 5 – A Practical Approach for Solving Parallel Runways Departure Sequencing Problem with Two Types of Aircrafts

5.1 Introduction

In this chapter, the problem of two parallel runways and two departing aircraft types is considered. Similar to the single runway problem, resequencing the aircraft based on type can improve the output. However, when two runways are available, an assignment to runway is now added to the sequencing problem.

Because two runways are available to choose from, the ATC must assign each departing aircraft to one of the runways. Many assignment options can be considered leading to many possible sequences.

In the next section, a proposed model is introduced, followed by a solution approach for small instances and then extended for large instances in section 5.3. Section 5.4 includes a computational study, while section 5.5 concludes the chapter.

5.2 Scheduling Model and Problem Description

Similar to the single runway problem in Chapter 3, a machine scheduling model can be used for the two-parallel runway problem. The runways can be represented as two identical parallel machines while the aircraft can be represented as jobs to be processed on the machines. Using the same notation as in section 3.2.1, the problem can be described as follows.

There are N jobs. There are two identical parallel machines. Each job i has a time that it is released, $r_{i,}$ to the production floor for processing. There are two types of jobs, type 1 and type 2. If a type 1 job precedes a type 2 job in any machine, there is a setup time S_{21} added after the processing of the type 1 job. If a type 2 job precedes a type 1 job, there is no setup time. There is no setup time between jobs of the same type. The processing time for all jobs is a constant p on either machine. The objective is to identify schedules that minimize C_{max} and F_{max}. Since the single machine version of this problem is NP-hard, the two identical parallel machines problem is also NP-Hard.

Because there are two machines, there are two decisions for each job required to optimize the objective: machine (runway) assignment and sequence position on the assigned machine

(runway). The machine assignment for each job is required first before sequencing jobs in each machine.

Following the scheduling formulation $\alpha/\beta/\gamma$ as described in (Pinedo 2008) where α represents the machine environment, β represents the processing characteristics and constraints, and γ represents the objective to be minimized, the problem can be described as:

$2P/\{r_i, S_{21}, p_i = p\}/\{C_{max}, F_{max}\}$, where *2P* refers to the two identical parallel machines environment.

5.3 Solution Approach for the Two A/C Types – Two Runways Problem

In order to solve the described scheduling problem in section 5.2 with the objective to minimize C_{max} or F_{max}, an exact algorithm is considered. The main idea of the algorithm is that once the assignment decisions are made, there are two independent machine sequencing sub-problems. Each sub-problem is the two aircraft types - single runway problem described in Chapter 3 and so the sub-problems can be solved using the proposed algorithm in Chapter 3. From there, it is guaranteed that if all possible sub-problem pairs (assignments) are identified, the optimal solution for minimizing C_{max} or F_{max} is within the identified solutions.

While it is practical to solve the two types - two runways problem for small instances (up to 15 jobs) using the proposed approach, it is not practical for large case. Thus, a modification to the approach is introduced. Section 5.3.1 describes the approach for small instances, while section 5.3.2 describes the modified approach for dynamic large instances.

5.3.1 Solution Approach for Small Instances

The two parallel identical machines problem $(2P/\{r_i, S_{21}, p_i = p\}/\{C_{max}, F_{max}\})$, can be divided into two single machine sub-problems by assigning some jobs to the first machine and the rest of the jobs to the other machine. In this case, each sub-problem is the independent single machine problem that can be solved using the approach described in Chapter 3. This means if all possible sub-problems pair are identified, then, it is possible to find the optimal solution for C_{max} and F_{max} for the two identical parallel machine problem. This can be achieved by solving each single machine sub-problem optimally. Then all possible machine assignments can be evaluated to find the optimal solution for the two machine problem

Counting all possible job assignments for all sub-problems pair can be done as follows:

Let N = total number of jobs, X = number of jobs to be assigned to the first machine, Y= number jobs to be assigned to the second machine = $N - X$. Then, the number of possible sub-problems pair = $\sum_{X=1}^{N-1} \binom{N}{X} = 2^N - 2$

Table 19 shows the number of possible pairs for N jobs.

Table 19: Number of Possible Pairs for *N* Jobs

No of jobs	No of pairs	No of jobs	No of pairs
3	6	12	4,094
4	14	13	8,190
5	30	14	16,382
6	62	15	32,766
7	126	16	65,534
8	254	17	131,070
9	510	18	262,142
10	1,022	19	524,286
11	2,046	20	1,048,574

Similar to the single machine problem, using this approach works for optimizing C_{max} or F_{max} for the two machine problem. Finding optimal C_{max} for all possible pairs will lead to finding the optimal C_{max} for the whole problem. The same argument holds for F_{max}.

When solving for C_{max}, there may be multiple optimal solutions for C_{max}. In this case, a better candidate would be the one with less runway usage. Runway usage can be defined as total processing time plus total setup (separation) time (i.e. idle time is excluded). Having lower runway usage means more idle time, thus, more room for additional flights in the future. it can be considered as a tie breaker in this case. For F_{max}, the average flow time (F_{avg}) is a reasonable tie breaker.

The algorithm for small instances can be described as follows:

1- Identify all possible pairs of single machine sub-problems that represent the two machine problem. This is done by identifying all possible job assignments for all sub-problems.

2- For each candidate pair, use the single machine solution approach to solve each sub-problem for C_{max} or F_{max}. C_{max} for each pair is the maximum C_{max} of the two sub-problems forming the pair. Similarly, F_{max} for each pair is the maximum F_{max} of the two sub-problems forming the pair.
3- Among all possible candidate pairs, pick the optimal one. In case of ties, runway usage is used as a tie breaker for C_{max}, while the average flow time of all aircraft (F_{avg}) is used as tie breaker for F_{max}.

As shown in Table 19 above, the number of possible sub-problem pairs increases exponentially as the number of jobs increase. Thus, using this algorithm is practical only for small instances (solvable in few seconds for up to 14 jobs). An extension to the algorithm to solve dynamic large instances is introduced next.

5.3.2 Extending the Approach to Solve Large Instances

In order to develop an algorithm to solve the problem for large instances, there are some considerations that make the algorithm efficient and fast. First, since the release time of the jobs are spread over the time horizon, it is possible to divide the problem into small sets. Each set can be solved separately. Then, considering the initial conditions that forms from the previous set, the whole problem can be solved by solving the small sets collectively. In this case, using the small instances algorithm (to solve the small sets) solves the large cases for both C_{max} and F_{max}. Since the CPU time for solving small instances depends on the number of jobs considered in the set, it is desirable to fix the number of jobs in each set to have a better estimation for the CPU time for the large case.

Another consideration to improve performance is to fix only the first job in each runway when solving each set, while carrying the remaining jobs into the next set. This ensure maximum flexibility for rescheduling in order to have best overall performance for the problem.

From the above points, an approach that achieves best performance in reasonable execution time for large instances can be described as following:

1- Order all jobs by release time. Consider the first K jobs for the first set.
2- Solve the set using small instances algorithm (Section 5.3.1).

3- From this set, commit only the first scheduled job on each runway.
4- Move to the next set, by considering the remaining *K-2* jobs from the previous set plus the next two jobs in release time order. Consider the initial conditions of each runway, based on the sequenced jobs, for the next set.
5- Repeat steps 2 to 4 until all jobs are sequenced.

5.4 Computational Study

To test the proposed methods against the typical (FCFS) method for scheduling departing aircrafts of two types using two runways, a computational experiment using Atlanta airport (ATL) data is performed.

ATL is the busiest airport in the United States in terms of aircraft movement and passengers. In 2016, the number of aircraft movements were approximately 898,000 and the total number of passengers using the airport was approximately 104M. The average number of daily scheduled departing flights was 1230, distributed during the day from 5am to 11pm. There are up to 100 scheduled aircraft per hour departures during weekdays (see Figure 13).

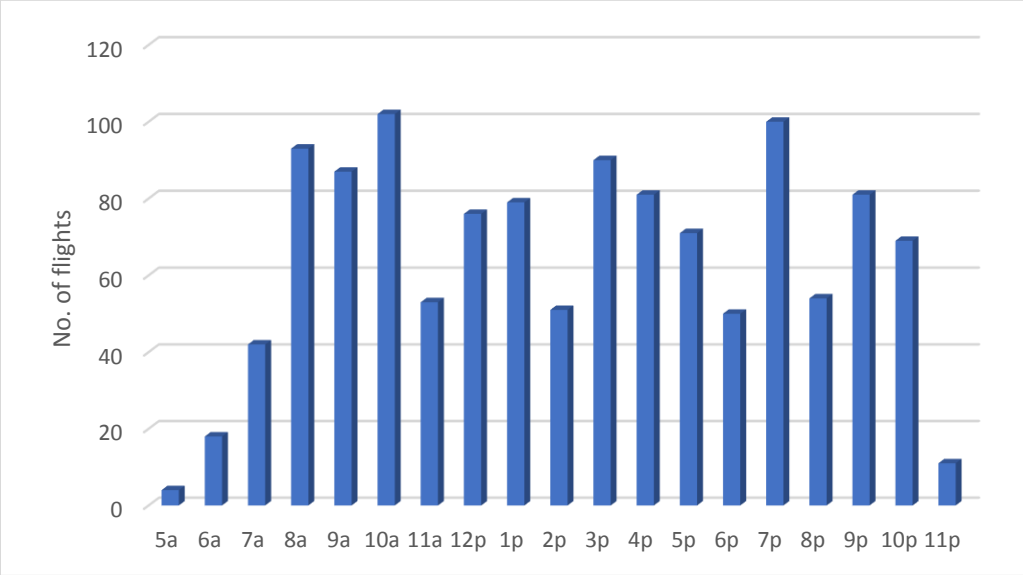

Figure 13: Number of Departing Flights Every Hour at ATL

For the experimental study, the data was collected from Atlanta airport daily schedule from 8:10 am to 9:05am for 100 aircraft scheduled to depart. There are two types of aircraft: large

(type 1) and heavy (type 2). Two runways are used for departure. We assume all departures are on time, taxi time is the same to each runway (thus not considered).

Three different scenarios are considered to examine the performance of the solution approaches:

1) Actual data (the ratio for type 1 to type 2 aircrafts is 50:1)
2) Same number of jobs with the same schedule time as actual data, but a ratio of 3:1 for the types (i.e. there are 3 type 1 aircraft for each type 2 aircraft).
3) Same number of jobs with the same schedule time as actual data, but a ratio of 1:1 for the types (i.e. there are 1 type 1 aircraft for each type 2 aircraft).

The details of the data (schedule times and types) are provided in Appendix D. The performance measures are throughput (C_{max}), runway utilization (*usage*), maximum delay (F_{max}) and average delay (F_{avg}). CPU time is also given for each objective.

Three different approaches to solve the problem:

1) FCFS with balanced assignment to runways
2) C_{max}* approach: the extended method focusing on minimizing C_{max} (and *Usage*)
3) F_{max}* approach: the extended method focusing on minimizing F_{max}.

Different set sizes are considered in the second and third approaches to measure the quality of each approach. The experiments were conducted using MATLAB (see Appendix G for codes). The summary of results is in Tables 20, 21 and 22. All measures are in minutes except CPU time which is in seconds. In these tables, the columns are defined as follows:

(C_{max} *x*) = completion time of all aircraft using runway *x*. *(x = {1,2})*
(C_{max}) = maximum completion time for both runways *(max {C_{max} 1, C_{max} 2})*
(*Usage x*) = total process times + separation times at runway *x* (i.e. idle time excluded)
(*Usage %*) = *(Usage 1 + Usage 2) / (C_{max} 1 + C_{max} 2)*
(F_{max}) = Maximum flow time of an aircraft in both runways
(F_{avg}) = Average flow time of all aircraft (Used as tie breaker for C_{max}* and F_{max}* approaches)
(CPU) = Computation time (in seconds)

Table 20: Experiment Summary – Part (1): Actual Data (N = 100)

Method	C_{max} 1	C_{max} 2	C_{max}	*Usage 1*	*Usage 2*	*Usage %*	F_{max}	F_{avg}	CPU
FCFS	59	59	59	51	51	86.44	8	4.45	0.04
C_{max}* Set size= 4	59	59	59	51	51	86.44	31	4.17	0.36
C_{max}* Set size= 6	59	59	59	50	52	86.44	31	4.17	1.38
C_{max}* Set size= 8	59	59	59	50	52	86.44	31	4.17	3.59
C_{max}* Set size= 10	59	59	59	50	52	86.44	31	4.17	19.32
C_{max}* Set size= 12	59	59	59	50	52	86.44	31	4.17	52.17
C_{max}* Set size=14	59	59	59	50	52	86.44	31	4.17	257.14
F_{max}* Set size= 4	59	59	59	51	51	86.44	8	4.45	0.40
F_{max}* Set size= 6	59	59	59	51	51	86.44	8	4.44	1.45
F_{max}* Set size= 8	59	59	59	51	51	86.44	8	4.44	5.93
F_{max}* Set size=10	59	59	59	51	51	86.44	8	4.44	17.33
F_{max}* Set size=12	59	59	59	50	52	86.44	8	4.41	72.13

Table 21: Experiment Summary – Part (2): Actual Data with Type Ratio of 3:1

Method	C_{max} 1	C_{max} 2	C_{max}	*Usage 1*	*Usage 2*	*Usage %*	F_{max}	F_{avg}	CPU
FCFS	62	62	62	59	60	95.97	11	6.8	0.02
C_{max}* Set size= 4	60	60	60	52	56	90.00	11	4.94	0.37
C_{max}* Set size= 6	60	59	60	52	55	89.92	13	4.90	1.09
C_{max}* Set size= 8	60	59	60	50	55	88.24	14	4.50	3.47
C_{max}* Set size= 10	59	59	59	50	54	88.14	18	4.44	12.05
C_{max}* Set size= 12	59	59	59	50	54	88.14	17	4.40	53.18
C_{max}* Set size=14	59	59	59	50	54	88.14	17	4.40	179.11
F_{max}* Set size= 4	59	60	60	54	55	91.60	9	5.33	0.37
F_{max}* Set size= 6	60	60	60	55	55	91.67	9	5.34	2.99
F_{max}* Set size= 8	59	60	60	54	55	91.60	9	5.24	12.36
F_{max}* Set size=10	59	60	60	51	55	89.08	8	4.67	75.25
F_{max}* Set size=12	59	60	60	51	55	89.08	8	4.66	453.97

Table 22: Experiment Summary – Part (3): Actual Data with Type Ratio of 1:1

Method	C_{max} 1	C_{max} 2	C_{max}	Usage 1	Usage 2	Usage %	F_{max}	F_{avg}	CPU
FCFS	63	63	63	63	62	99.21	14	7.76	0.02
C_{max}* Set size= 4	60	60	60	54	53	89.17	14	4.85	0.17
C_{max}* Set size= 6	59	60	60	51	54	88.24	11	4.58	0.85
C_{max}* Set size= 8	59	60	60	51	54	88.24	11	4.59	2.72
C_{max}* Set size= 10	60	59	60	50	54	87.39	12	4.50	7.29
C_{max}* Set size= 12	60	59	60	50	54	87.39	12	4.50	30.79
C_{max}* Set size=14	60	59	60	50	54	87.39	12	4.50	134.02
F_{max}* Set size= 4	61	60	61	55	55	90.91	9	5.38	0.35
F_{max}* Set size= 6	60	60	60	53	56	90.83	9	5.08	3.21
F_{max}* Set size= 8	60	60	60	53	55	90.00	9	4.95	11.75
F_{max}* Set size=10	59	60	60	52	55	89.92	9	4.90	66.28
F_{max}* Set size=12	59	60	60	53	54	89.92	9	4.81	478.07

In the first case, Table 20, where the number of type 2 aircraft is very small and release times scattered throughout the time horizon, the total runway usage did not improve when using the approaches to minimize F_{max} and C_{max} compared to FCFS. The maximum delay (F_{max}) increased from 8 minutes (using FCFS) to 31 minutes when minimizing $C_{max,}$ and did not improve when minimizing F_{max}.

In the second case, Table 21, with a type ratio of 1:3, the runway usage improved from 119 minutes (using FCFS) to 106 minutes when minimizing $F_{max,}$ and to 104 minutes when minimizing C_{max}, the maximum delay (F_{max}) improved from 11 minutes (using FCFS) to 8 minutes when minimizing $F_{max,}$ but increased to 17 minutes when minimizing $C_{max.}$

In the third case, Table 22, with a type ratio of 1:1, the runway usage improved from 125 minutes (using FCFS) to 107 minutes when minimizing $F_{max,}$ and to 104 minutes when minimizing C_{max}, the maximum delay (F_{max}) improved from 14 minutes (using FCFS) to 9 minutes when minimizing $F_{max,}$ and to 12 minutes when minimizing $C_{max.}$ Figure 14 and 15 summarize the results for all cases.

For C_{max}* and F_{max}* approaches, as the number of jobs considered in each set increases, the performance of usage and F_{max} improves, respectively. However, CPU time also increases as set size increases.

Figures 16 and 17 show the relation between set size and values of C_{max}, F_{max}, and CPU time, when using C_{max}* and F_{max}* approaches in the third case, respectively. From the figures, we can estimate that the approaches are limited to practically solve instances of up to 12 jobs per set. However, using these approaches to solve sets for up to 12 jobs is reasonable to solve the ATP problem as results from experiment has shown. Note that C_{max}* approach needs less CPU time compared to F_{max}* approach because the solution approach used in each sub-problem is a constructive as described in Chapter 3.

When the diversity of types ratio is high and release times are condensed (as in the second and third cases), it is possible to have dominant solutions curve between F_{max} and *usage* (Figure 18 and 19 shows the dominant solutions for second and third cases, respectively). This can provide the decision makers (control tower) an efficient tool to maximizing one of the measures or balance the output between both as desired. If it is more important to prioritize throughput over maximum delay, then, solving the problem using C_{max}* approach will be the desired choice, and vice versa.

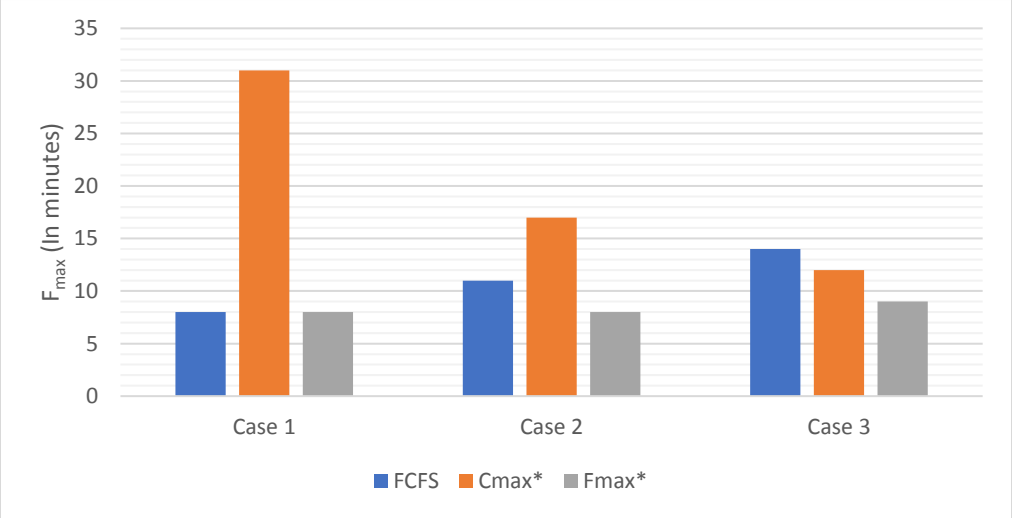

Figure 14: Comparing F_{max} Performance Using FCFS, F_{max}* and C_{max}* Approaches

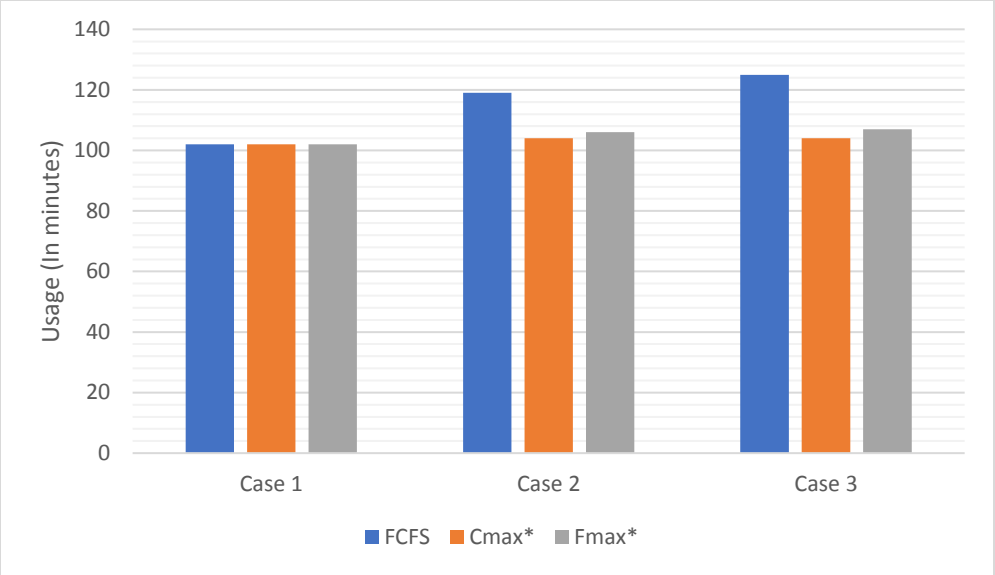

Figure 15: Comparing *Usage* Performance using FCFS, F_{max}* and C_{max}* Approaches

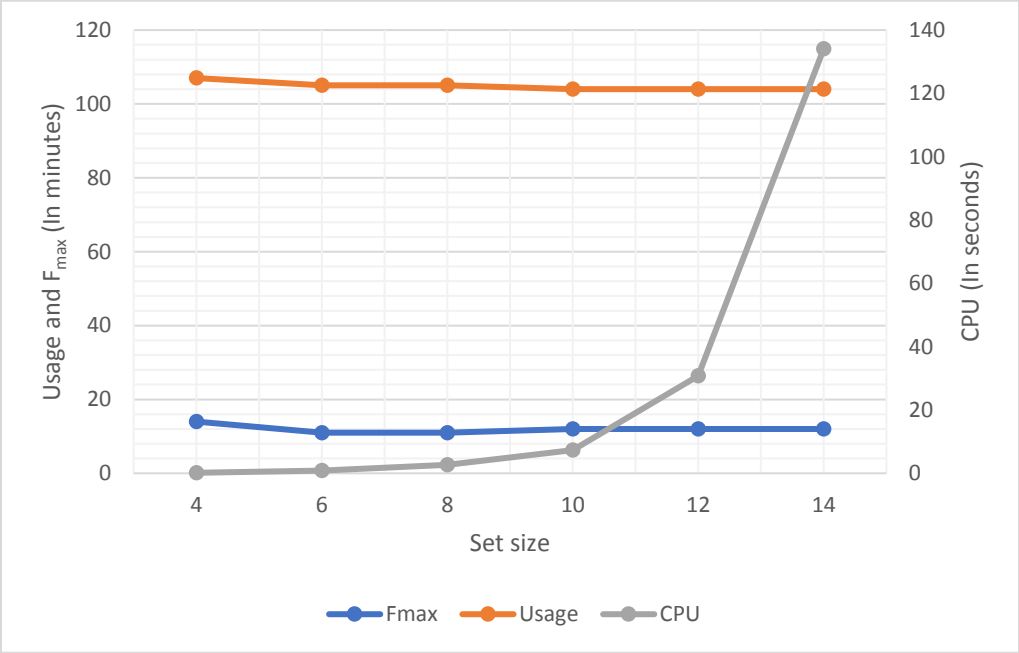

Figure 16: C_{max}* Approach Performance vs Set Size for Case 3

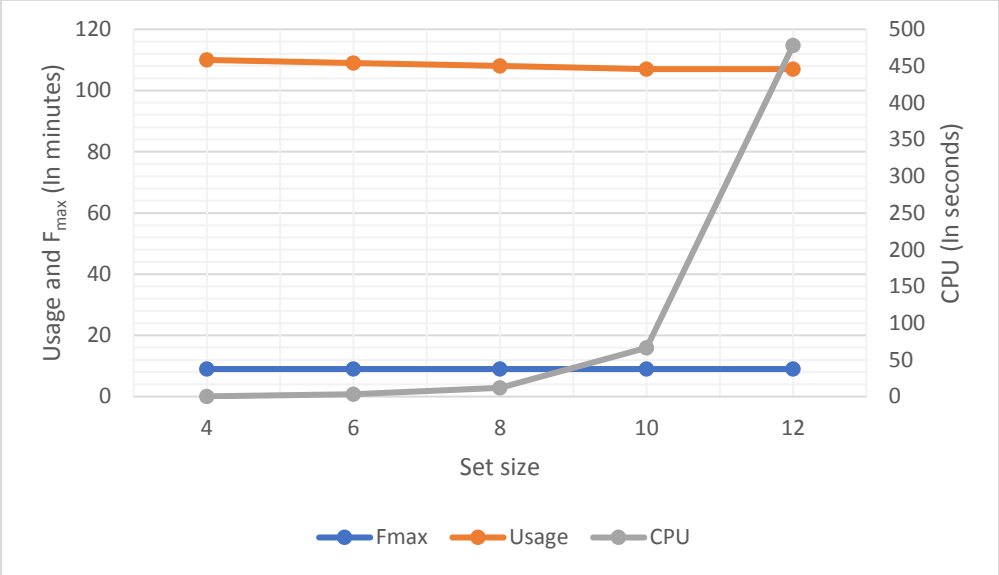

Figure 17: F_{max}* Approach Performance vs Set Size for Case 3

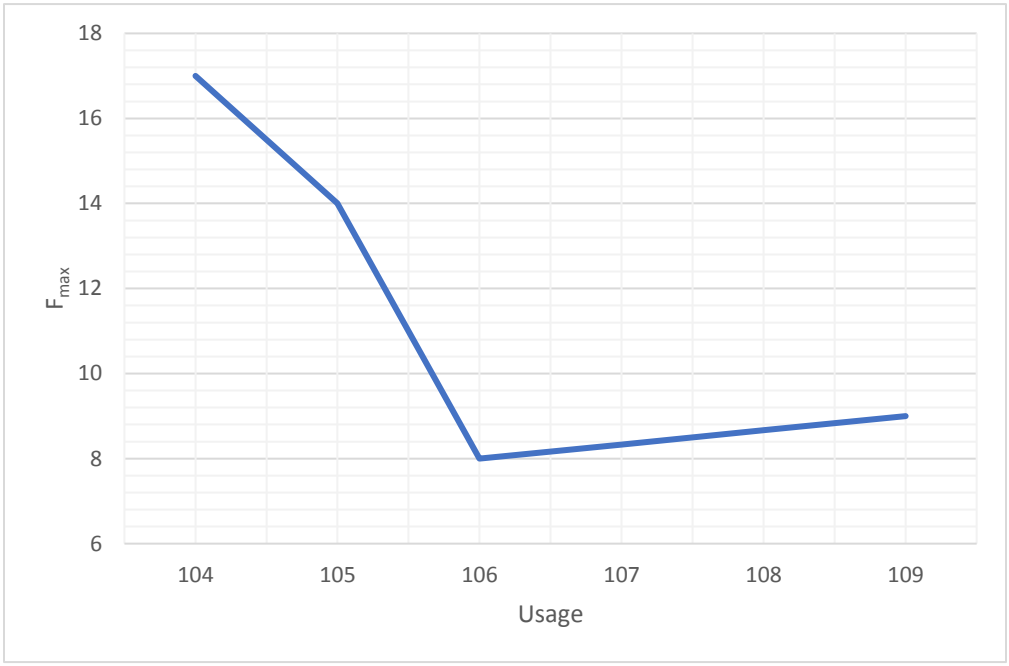

Figure 18: Dominant solutions for *Usage* and F_{max} (Case 2)

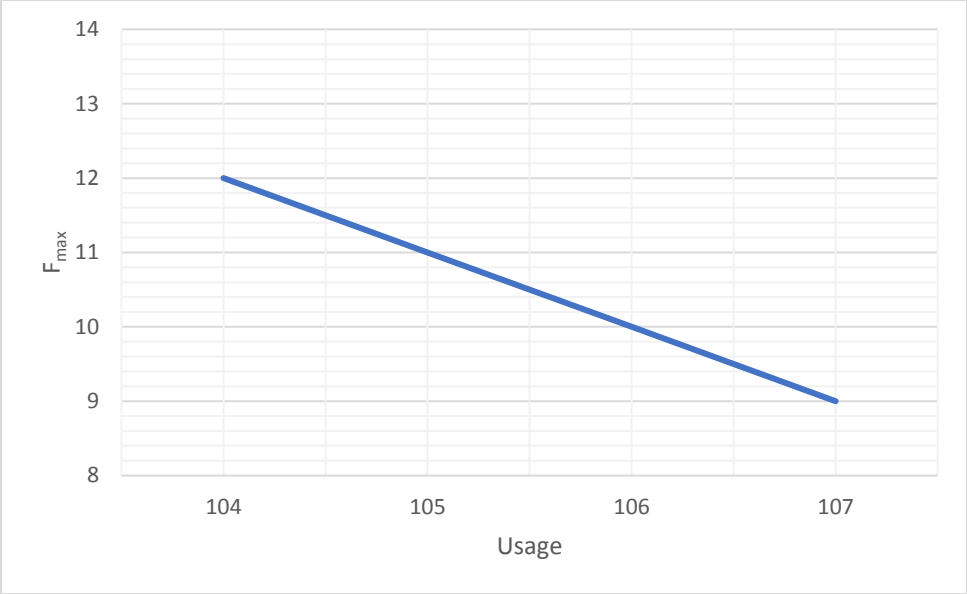

Figure 19: Dominant solutions for *Usage* and F_{max} (Case 3)

5.5 Conclusion

The goal in this chapter is to provide an approach that can solve the problem of scheduling two types of departing aircrafts on two parallel runways effectively and with flexibility. Effectiveness is achieved by solving the problem in few seconds for a large real case (such as 100 jobs) with efficient results for multiple performance measures. Flexibility is achieved by providing two approaches for multiple performance measures, C_{max} and F_{max}.

As shown in the computational experiments, both approaches were effective. Both approaches can solve the problem practically in few seconds to provide better output compared to FCFS for a real case such as the Atlanta airport. Each approach provided better schedules in terms of both of the performance measures (C_{max} or F_{max}) compared to FCFS. In addition, both approaches perform better with a diverse type ratio, as compared to FCFS.

The main contribution in this chapter is introducing an assignment approach to deal with the runway assignment decision and integrating it with the sequencing approach that is used to optimize performance at each runway.

Chapter 6 – Future Work

There are four topics considered for future work: 1- Mixed operations problem (departure/arrival) for 2 runways / 2 types, 2- Different taxi times from gate to each runway (for 2 runways / 2 types), 3- Introducing Additional objectives, and 4- Issue of taxiway passing as occurs at the Atlanta airport. Each is described below.

1- Mixed Operations Problem (Departure/Arrival) for 2 Runways/2 A/C Types

From Table 23, considering mixed mode may improve runway performance (minimizing maximum delay and usage). A departure followed by an arrival, or an arrival followed by a departure has minimum separation time (0 units). However, there will be 4 additional separation times to consider in the problem. This will add considerable complexity to the problem. This may be an important extension for smaller, but busy, airports (JFK, for example).

Table 23: Minimum Separation Time (in Minutes)

Leading / Following		Arrival		Departure	
		Heavy	Large	Heavy	Large
Arrival	Heavy	1.5	2.25	0	0
	Large	1.25	1.5	0	0
Departure	Heavy	0	0	0	1
	Large	0	0	0	0

2- Different Taxi Times from Gate to Each Runway (for 2 Runways / 2 Types)

When the taxi-time is the same from the gate to either runway, the approach of sequencing jobs that are completed, in the final solution before the release of new jobs is efficient. If the taxi time is significantly different to each runway, it is possible for a job to be completed on one runway while it might not be ready to take-off if it used the other runway. Assigning the job to the second runway may improve the overall solution. Thus, fixing the job in the first runway might not be efficient for the overall solution. Using "completion time of job < release time of incoming job" rule is not efficient in this case. (i.e. fixing a completed job on one runway might not improve the overall solution if it can still be considered an option for the other runway). This is an extension that can be considered for future work.

3- Introducing Additional Objectives

Other objectives might be considered for the ATP. While maximizing throughput and minimizing maximum delay are primary desired objectives, other goals such as minimizing weighted delays might be considered in the problem. Adding more objectives to the current efficient frontier is considered for the future.

4- Issue of taxiway passing as in Atlanta airport

Some airports, such as the Atlanta airport, have the capability, due to its layout, of allowing aircraft to pass one another on the taxi ways. This allows a larger number of A/C to be considered in the sequencing. This issue can be considered to provide better solution approach for such airport environment.

REFERENCES

Abela, J., Abramson, D., Krishnamoorthy, M., De Silva, A., & Mills, G. (1993, July). Computing optimal schedules for landing aircraft. In *Proceedings of the 12th National ASOR Conference* (pp. 71-90).

Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research, 246*(2), 345-378.

Allahverdi, A., Gupta, J. N., & Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *Omega*, *27*(2), 219-239.

Allahverdi, A., Ng, C. T., Cheng, T. E., & Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, *187*(3), 985-1032.

Anagnostakis, I., & Clarke, J. P. (2003, January). Runway operations planning: a two-stage solution methodology. In *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on* (pp. 12-pp). IEEE.

Anagnostakis, I., Clarke, J. P., Bohme, D., & Volckers, U. (2001, January). Runway operations planning and control sequencing and scheduling. In *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on* (pp. 12-pp). IEEE.

Asano, M., & Ohta, H. (1996). Single machine scheduling using dominance relation to minimize earliness subject to ready and due times. *International Journal of Production Economics*, *44*(1), 35-43.

Asano, M., & Ohta, H. (1999). Scheduling with shutdowns and sequence dependent set-up times. *International Journal of Production Research, 37*(7), 1661-1676.

Atkin, J. A., Burke, E. K., Greenwood, J. S., & Reeson, D. (2007). Hybrid metaheuristics to aid runway scheduling at London Heathrow airport. *Transportation Science*, *41*(1), 90-106.

Avella, P., Boccia, M., Mannino, C., & Vasilyev, I. (2017). Time-Indexed Formulations for the Runway Scheduling Problem. *Transportation Science*.

Balakrishnan, H., & Chandran, B. (2007, July). Efficient and equitable departure scheduling in real-time: new approaches to old problems. In *7th USA-Europe Air Traffic Management Research and Development Seminar* (pp. 02-05).

Balakrishnan, H., & Chandran, B. G. (2010). Algorithms for scheduling runway operations under constrained position shifting. *Operations Research*, *58*(6), 1650-1665.

Baptiste, P. (2000). Batching identical jobs. *Mathematical methods of operations research*, *52*(3), 355-367.

Baptiste, P., & Le Pape, C. (2005). Scheduling a single machine to minimize a regular objective function under setup constraints. *Discrete optimization*, *2*(1), 83-99.

Bäuerle, N., Engelhardt-Funke, O., & Kolonko, M. (2007). On the waiting time of arriving aircrafts and the capacity of airports with one or two runways. *European Journal of Operational Research*, *177*(2), 1180-1196.

Bayen, A. M., Tomlin, C. J., Ye, Y., & Zhang, J. (2004, December). An approximation algorithm for scheduling aircraft with holding time. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on* (Vol. 3, pp. 2760-2767). IEEE.

Beasley, J. E., Krishnamoorthy, M., Sharaiha, Y. M., & Abramson, D. (2000). Scheduling aircraft landings—the static case. *Transportation science*, *34*(2), 180-197.

Beasley, J. E., Krishnamoorthy, M., Sharaiha, Y. M., & Abramson, D. (2004). Displacement problem and dynamically scheduling aircraft landings. *Journal of the operational research society*, *55*(1), 54-64.

Beasley, J. E., Sonander, J., & Havelock, P. (2001). Scheduling aircraft landings at London Heathrow using a population heuristic. *Journal of the operational Research Society*, *52*(5), 483-493.

Bencheikh, G., Boukachour, J., Alaoui, A. E. H., & Khoukhi, F. E. (2009). Hybrid method for aircraft landing scheduling based on a job shop formulation. *International Journal of Computer Science and Network Security*, *9*(8), 78-88.

Bennell, J. A., Mesgarpour, M., & Potts, C. N. (2013). Airport runway scheduling. *Annals of Operations Research*, *204*(1), 249-270.

Bennell, J. A., Mesgarpour, M., & Potts, C. N. (2016). Dynamic scheduling of aircraft landings. *European Journal of Operational Research*.

Bianco, L., Dell'Olmo, P., & Giordani, S. (2006). Scheduling models for air traffic control in terminal areas. *Journal of Scheduling*, *9*(3), 223-253.

Bianco, L., Dell'Olmo, P., & Giordani, S. (1999). Minimizing total completion time subject to release dates and sequence-dependentprocessing times. *Annals of Operations Research*, *86*, 393-415.

Bianco, L., Ricciardelli, S., Rinaldi, G., & Sassano, A. (1988). Scheduling tasks with sequence-dependent processing times. *Naval Research Logistics (NRL)*, *35*(2), 177-184.

Brentnall, A. R. (2006). Aircraft arrival management. PhD thesis, University of Southampton, UK

Brentnall, A. R., & Cheng, R. C. H. (2009). Some effects of aircraft arrival sequence algorithms. *Journal of the Operational Research Society*, *60*(7), 962-972.

Brinton, C. R. (1992, October). An implicit enumeration algorithm for arrival aircraft. In *Digital Avionics Systems Conference, 1992. Proceedings., IEEE/AIAA 11th* (pp. 268-274). IEEE.

Caprì, S., & Ignaccolo, M. (2004). Genetic algorithms for solving the aircraft-sequencing problem: the introduction of departures into the dynamic model. *Journal of Air Transport Management*, *10*(5), 345-351.

Cheng, V. H. L., Crawford, L. S., & Menon, P. K. (1999). Air traffic control using genetic search techniques. In *Control Applications, 1999. Proceedings of the 1999 IEEE International Conference on* (Vol. 1, pp. 249-254). IEEE.

Chou, F. D., Wang, H. M., & Chang, T. Y. (2009). Algorithms for the single machine total weighted completion time scheduling problem with release times and sequence-dependent setups. *The International Journal of Advanced Manufacturing Technology*, *43*(7-8), 810-821.

Ciesielski, V., & Scerri, P. (1998, May). Real time genetic scheduling of aircraft landing times. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on* (pp. 360-364). IEEE.

Clare, G., & Richards, A. G. (2011). Optimization of taxiway routing and runway scheduling. *IEEE Transactions on Intelligent Transportation Systems*, *12*(4), 1000-1013.

Craig, A., Ketzscer, R., Leese, R. A., Noble, S. D., Parrott, K., Preater, J., ... & Wood, D. A. (2001). The sequencing of aircraft departures. *40th European study group with industry, Keele*.

De Neufville, R. (2016). Airport systems planning and design. *Air Transport Management: An International Perspective*, 61.

Dear, R. G., & Sherif, Y. S. (1989). The dynamic scheduling of aircraft in high density terminal areas. *Microelectronics Reliability*, *29*(5), 743-749.

Ernst, A. T., Krishnamoorthy, M., & Storer, R. H. (1999). Heuristic and exact algorithms for scheduling aircraft landings. *Networks*, *34*(3), 229-241.

Farn, C. K., & Muhlemann, A. P. (1979). The dynamic aspects of a production scheduling problem. *International Journal of Production Research*, *17*(1), 15-21.

Feng, X. R., Feng, X. J., & Liu, D. (2013). The application of Ant colony optimization algorithm in the flight landing scheduling problem. In *Applied Mechanics and Materials* (Vol. 411, pp. 2698-2703). Trans Tech Publications.

Feng, X., Feng, X., & Wang, X. (2016). An ant colony optimisation method based on pruning technique for the aircraft arrival sequencing and scheduling problem. *International Journal of Applied Decision Sciences*, *9*(4), 333-347.

Furini, F., Kidd, M. P., Persiani, C. A., & Toth, P. (2015). Improved rolling horizon approaches to the aircraft sequencing problem. *Journal of Scheduling*, *18*(5), 435-447.

Furini, F., Persiani, C. A., & Toth, P. (2012, April). Aircraft sequencing problems via a rolling horizon algorithm. In *International Symposium on Combinatorial Optimization* (pp. 273-284). Springer Berlin Heidelberg.

Ghoniem, A., & Farhadi, F. (2015). A column generation approach for aircraft sequencing problems: a computational study. *Journal of the Operational Research Society*, *66*(10), 1717-1729.

Ghoniem, A., Farhadi, F., & Reihaneh, M. (2015). An accelerated branch-and-price algorithm for multiple-runway aircraft sequencing problems. *European Journal of Operational Research*, *246*(1), 34-43.

Ghoniem, A., Sherali, H. D., & Baik, H. (2014). Enhanced models for a mixed arrival-departure aircraft sequencing problem. *INFORMS Journal on Computing*, *26*(3), 514-530.

Guépet, J., Briant, O., Gayon, J. P., & Acuna-Agost, R. (2017). Integration of aircraft ground movements and runway operations. *Transportation Research Part E: Logistics and Transportation Review*, *104*, 131-149.

Gupta, G., Malik, W., & Jung, Y. C. (2009, September). A mixed integer linear program for airport departure scheduling. In *9th AIAA aviation technology, integration, and operations conference (ATIO)* (pp. 21-23).

Hancerliogullari, G., Rabadi, G., Al-Salem, A. H., & Kharbeche, M. (2013). Greedy algorithms and metaheuristics for a multiple runway combined arrival-departure aircraft sequencing problem. *Journal of Air Transport Management*, *32*, 39-48.

Hansen, J. V. (2004). Genetic search methods in air traffic control. *Computers & Operations Research*, *31*(3), 445-459.

Hu, X. B., & Chen, W. H. (2005). Receding horizon control for aircraft arrival sequencing and scheduling. *IEEE Transactions on Intelligent Transportation Systems*, *6*(2), 189-197.

Idris, H. R., Delcaire, B., Anagnostakis, I., Hall, W. D., Clarke, J. P., Hansman, R. J., & Odoni, A. R. (1998, December). Observations of departure processes at Logan airport to support the development of departure planning tools. In *2nd USA/Europe air traffic management R&D seminar* (pp. 1-4).

Lee, H., & Balakrishnan, H. (2008, June). Fuel cost, delay and throughput tradeoffs in runway scheduling. In *2008 American Control Conference* (pp. 2449-2454). IEEE.

Lieder, A., Briskorn, D., & Stolletz, R. (2015). A dynamic programming approach for the aircraft landing problem with aircraft classes. *European Journal of Operational Research*, *243*(1), 61-69.

Ma, W., Xu, B., Liu, M., & Huang, H. (2014). An efficient approximation algorithm for aircraft arrival sequencing and scheduling problem. *Mathematical Problems in Engineering*, *2014*.

Ma, W., Xu, B., Liu, M., & Huang, H. (2016). An efficient algorithm based on sparse optimization for the aircraft departure scheduling problem. *Computational and Applied Mathematics*, *35*(2), 371-387.

Murça, M. C. R., & Müller, C. (2015). Control-based optimization approach for aircraft scheduling in a terminal area with alternative arrival routes. *Transportation research part E: logistics and transportation review*, *73*, 96-113.

Nogueira, T. H., de Carvalho, C. R. V., & Ravetti, M. G. (2014). Analysis of mixed integer programming formulations for single machine scheduling problems with sequence dependent setup times and release dates. *Optimization Online*.

Ovacik, I. M., & Uzsoy, R. (1994a). Exploiting shop floor status information to schedule complex job shops. *Journal of manufacturing systems*, *13*(2), 73-84.

Ovacikt, I. M., & Uzsoy, R. (1994b). Rolling horizon algorithms for a single-machine dynamic scheduling problem with sequence-dependent setup times. *The International Journal of Production Research*, *32*(6), 1243-1263.

Pinedo, M. L. (2008). Scheduling: Theory, Algorithms, and Systems.

Pinol, H., & Beasley, J. E. (2006). Scatter search and bionomic algorithms for the aircraft landing problem. *European Journal of Operational Research*, *171*(2), 439-462.

Potts, C. N., & Kovalyov, M. Y. (2000). Scheduling with batching: A review. *European journal of operational research*, *120*(2), 228-249.

Potts, C. N., & Van Wassenhove, L. N. (1992). Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity. *Journal of the Operational Research Society*, *43*(5), 395-406.

Psaraftis, H. N. (1978). *A dynamic programming approach to the aircraft sequencing problem*. Cambridge, Mass.: Massachusetts Institute of Technology, Flight Transportation Laboratory, [1978].

Psaraftis, H. N. (1980). A dynamic programming approach for sequencing groups of identical jobs. *Operations Research*, *28*(6), 1347-1359.

Pujet, N. (1999). *Modeling and control of the departure process of conjested airports* (Doctoral dissertation, Massachusetts Institute of Technology).

Raman, N., Rachamadugu, R. V., & Talbot, F. B. (1989). Real-time scheduling of an automated manufacturing center. *European Journal of Operational Research*, *40*(2), 222-242.

Randall, M. (2002). Scheduling aircraft landings using ant colony optimisation. In *6th IASTED International Conference Artificial Intelligence and Soft Computing, Banff, Alberta, Canada* (pp. 129-133).

Rojas-Santiago, M., Muthuswamy, S., Vélez-Gallego, M. C., & Montoya-Torres, J. R. (2014, January). Combining LR and 2-opt for scheduling a single machine subject to job ready times and sequence dependent setup times. In *Proceedings of IIE Annual Conference and Expo 2014* (pp. 92-100).

Sabar, N. R., & Kendall, G. (2015). An iterated local search with multiple perturbation operators and time varying perturbation strength for the aircraft landing problem. *Omega*, *56*, 88-98.

Samà, M., D'Ariano, A., & Pacciarelli, D. (2013). Rolling horizon approach for aircraft scheduling in the terminal control area of busy airports. *Transportation Research Part E: Logistics and Transportation Review*, *60*, 140-155.

Samà, M., D'Ariano, A., D'Ariano, P., & Pacciarelli, D. (2014). Optimal aircraft scheduling and routing at a terminal control area during disturbances. *Transportation Research Part C: Emerging Technologies*, *47*, 61-85.

Sama, M., D'Ariano, A., Toli, A., Pacciarelli, D., & Corman, F. (2015, June). A variable neighborhood search for optimal scheduling and routing of take-off and landing aircraft. In *Models and Technologies for Intelligent Transportation Systems (MT-ITS), 2015 International Conference on* (pp. 491-498). IEEE.

Shin, H. J., Kim, C. O., & Kim, S. S. (2002). A tabu search algorithm for single machine scheduling with release times, due dates, and sequence-dependent set-up times. *The International Journal of Advanced Manufacturing Technology*, *19*(11), 859-866.

Soomer, M. J., & Franx, G. J. (2008). Scheduling aircraft landings using airlines' preferences. *European Journal of Operational Research*, *190*(1), 277-291.

Soykan, B. (2016). A hybrid Tabu/Scatter Search algorithm for simulation-based optimization of multi-objective runway operations scheduling. Old Dominion University.

Stevens, G., & Stevens, S. G. (1995). An approach to scheduling aircraft landing times using genetic algorithms. Honours thesis, RMIT University, Melbourne, Australia.

Sun, X., Noble, J. S., & Klein, C. M. (1999). Single-machine scheduling with sequence dependent setup to minimize total weighted squared tardiness. *IIE transactions*, *31*(2), 113-124.

Trivizas, D. A. (1998). Optimal scheduling with maximum position shift (MPS) constraints: a runway scheduling application. *Journal of Navigation*, *51*(02), 250-266.

Uzsoy, R., Lee, C. Y., & Martin-Vega, L. A. (1992). Scheduling semiconductor test operations: minimizing maximum lateness and number of tardy jobs on a single machine. *Naval Research Logistics (NRL)*, *39*(3), 369-388.

Van Leeuwen, P., Hesselink, H., & Rohling, J. (2002). Scheduling aircraft using constraint satisfaction. *Electronic notes in theoretical computer science*, *76*, 252-268.

Vasilyev, I. L., Avella, P., & Boccia, M. (2016). A branch and cut heuristic for a runway scheduling problem. *Automation and Remote Control*, *77*(11), 1985-1993.

Veidal, E. (2007). *Scheduling Aircraft Landings-The Dynamic Case* (Doctoral dissertation, Technical University of Denmark, DTU, DK-2800 Kgs. Lyngby, Denmark).

Wang, C. S., & Uzsoy, R. (2002). A genetic algorithm to minimize maximum lateness on a batch processing machine. *Computers & Operations Research*, *29*(12), 1621-1640.

Wang, F., Zhang, J. F., Chen, Q., & Sui, D. (2015, July). Dynamic Arrival Scheduling Based on Branch and Bound Algorithm. In *15th COTA International Conference of Transportation Professionals*.

Webster, S., & Baker, K. R. (1995). Scheduling groups of jobs on a single machine. *Operations Research*, *43*(4), 692-703.

Wen, M., Larsen, J., & Clausen, J. (2005). *An exact algorithm for aircraft landing problem*. Technical University of Denmark

Xie, J., Zhou, Y., & Zheng, H. (2013). A hybrid metaheuristic for multiple runways aircraft landing problem based on bat algorithm. *Journal of Applied Mathematics*, *2013*.

Xu, B. (2017). An efficient Ant Colony algorithm based on wake-vortex modeling method for aircraft scheduling problem. *Journal of Computational and Applied Mathematics*, *317*, 157-170.

Yu, S. P., Cao, X. B., & Zhang, J. (2011). A real-time schedule method for Aircraft Landing Scheduling problem based on Cellular Automation. *Applied Soft Computing*, *11*(4), 3485-3493.

Yuan, J. J., Yang, A. F., & Cheng, T. E. (2004). A note on the single machine serial batching scheduling problem to minimize maximum lateness with identical processing times. *European journal of operational research*, *158*(2), 525-528.

Zhang, X., & Xie, Q. (2015). Single Machine Group Scheduling with Position Dependent Processing Times and Ready Times. *Mathematical Problems in Engineering, 2015*.

Zhou, H., & Jiang, X. (2015). Multirunway optimization schedule of airport based on improved genetic algorithm by dynamical time window. *Mathematical Problems in Engineering, 2015*.

APPENDICES

Appendix A: Airport Traffic Summary

Table 24: Airport Traffic Summary for Top 30 Busiest Airports in USA (2015)

Ranking	City / Airport Code	Total Passengers	Aircraft Movements	# Active runways
1	Atlanta GA (ATL)	101,491,106	882,497	5
2	Chicago IL (ORD)	76,949,504	875,136	8
3	Los Angeles CA (LAX)	74,937,004	655,564	4
4	Dallas/Fort Worth TX (DFW)	64,074,762	681,261	7
5	New York NY (JFK)	56,827,154	438,897	4
6	Denver CO (DEN)	54,014,502	541,213	6
7	San Francisco CA (SFO)	50,057,887	429,815	4
8	Las Vegas NV (LAS)	45,443,900	530,330	4
9	Charlotte NC (CLT)	44,876,627	543,944	4
10	Miami FL (MIA)	44,350,247	412,915	4
11	Phoenix AZ (PHX)	44,003,840	440,411	3
12	Houston TX (IAH)	43,023,224	502,844	4
13	Seattle WA (SEA)	42,340,537	381,408	3
14	Orlando FL (MCO)	38,727,749	308,169	4
15	Newark NJ (EWR)	37,494,704	415,534	3
16	Minneapolis MN (MSP)	36,582,854	404,612	4
17	Boston MA (BOS)	33,515,905	372,930	6
18	Detroit MI (DTW)	33,440,112	379,376	6
19	Philadelphia PA (PHL)	31,444,403	411,368	4
20	New York NY (LGA)	28,437,668	360,274	2
21	Fort Lauderdale, FL (FLL)	26,941,671	278,002	2
22	Baltimore MD (BWI)	23,823,532	246,454	3
23	Washington DC (DCA)	23,012,191	292,781	3
24	Chicago IL (MDW)	22,221,499	253,519	5
25	Salt Lake City UT (SLC)	22,152,498	311,857	4
26	Washington DC (IAD)	21,498,902	269,070	4
27	San Diego CA (SAN)	20,081,258	194,215	1
28	Honolulu HI (HNL)	19,869,707	312,655	4
29	Tampa FL (TPA)	18,815,425	189,865	3
30	Portland OR (PDX)	16,850,952	218,021	3

(Source: Airports Council International)

Appendix B: Single Machine Scheduling Papers Summary

Summary for papers that considered dependent setup times and release times for a single machine environment:

- Farn and Muhlemann (1979) is one of the early papers that considered having nonzero release times (dynamic environment) in the single machine scheduling problem with the goal of minimizing total setup time. Given sequence dependent setup times, they considered many heuristics and optimizing procedures to solve the problem in the dynamic case, and show that a heuristic that performed better in the dynamic case might not be the best heuristic for the static case (when all jobs are available immediately).
- Bianco et al. (1988) considered the problem of scheduling in a single machine to minimize the maximum completion time with the assumptions of no batches, non-preemptive, nonzero ready times, and sequence dependent setup times. They proposed upper bound, lower bound, and dominance criteria to be used in a branch and bound algorithm. They solved the problem optimally up to 20 jobs in a reasonable computation time.
- Raman et al. (1989) developed characteristics of the optimal solutions to the mean flow time and mean tardiness for a single machine problem with batch consideration. They used implicit enumeration to minimize the mean tardiness when dependent setup times and release times are present in the problem.
- Uzsoy et al. (1992) considered solving single machine scheduling to minimize maximum lateness in one case and the number of tardy jobs in another case. Sequence dependent setup times and release times with precedence constraints were assumed. They used a heuristic approach with a worst case bound for solving the problem approximately. A dynamic programming approach is also considered for the case when all jobs are available.
- Ovacik and Uzoy (1994a) minimized the maximum lateness for a job shop, subject to release time and sequence dependent setup time constraints. They used a heuristic approach for the problem in three different shop configurations. In (1994b) they considered a family of heuristics to solve the same problem for a single machine environment.

- Asano and Ohta (1996) minimized the total earliness subject to release times, due dates, sequence dependent setup times and no tardy jobs allowed. They solve the problem optimally using a dominance relation and a strong lower bound in a branch and bound algorithm.
- Asano and Ohta (1999) solved a single machine problem for minimizing maximum tardiness. They considered constraints such as release times, due date, sequence dependent setup times and shutdowns (disruptive events). A branch and bound procedure was developed to solve the problem.
- Sun et al. (1999) considered the single machine problem with release dates, due dates, sequence dependent setup times and no preemption allowed. They used Lagrangian relaxation to minimize the weighted sum of squared tardiness. The experimental results compared this approach with other heuristics such as local search, tabu search and simulated annealing and showed better solution quality.
- Baptiste (2000) solved a scheduling problem for multiple objectives with release dates, constant setup time, equal processing times, and batch availability (all jobs in the batch are processed simultaneously). Dynamic programming was used to find a polynomial time solution for two types of batching, serial and parallel batching.
- Shin et al. (2002) used tabu search to minimize the maximum lateness for a single machine problem with release times, due dates, and sequence dependent setup time constraints. They compared their algorithm performance with the method proposed by Ovacik and Uzsoy (1994b).
- Yuan et al. (2004) considered the single machine problem with serial batching, release times, constant setup time between batches and equal job processing times. They proposed an exact algorithm to solve the problem for L_{max} in polynomial CPU time.
- Baptiste and Le Pape (2005) considered the single machine problem with the constraints of release dates, deadlines, and setup times between families of jobs. They used branch and bound to minimize the sum of costs ($\sum f_i$), where f_i corresponds to completion time, C_i, of job i.
- Chou et al. (2009) developed two exact algorithms (constraint programming and branch-and-bound) for small instances and two heuristics for large instances to

minimize the total weighted completion time for a single machine with release times and sequence dependent setup times constraints.

- Nogueira et al. (2014) solved the single machine problem with release times and sequence dependent setup times constraints. They analyzed six different MIP models, based on four different paradigms for the single machine problem, to minimize the total weighted completion time and total weighted tardiness. Computational experiments were performed to compare the different formulations performance.
- Rojas-Santiago et al. (2014) considered minimizing the completion time for a single machine problem with release times and sequence dependent setup times constraints. They used Lagrangian relaxation (LR) for an initial solution and a 2-opt heuristic for the problem. Experimental evidence compared their method's performance with commercial solvers.
- Zhang and Xie (2015) solved the problem of a single machine constrained by release times, position dependent processing times and group length dependent setup time, with the assumption of group technology (i.e. no family can be split), to optimize C_{max}. They proved an optimal solution can be found if special conditions apply. They also solved a special case for the problem in polynomial time.

Appendix C: Data from JFK Airport for the Single Runway Experiment

Table 25: JFK Airport Data

#	Schedule time (minutes)	Modified scheduled time	Aircraft type		
			2 types case	3 types 1st case	3 types 2nd case
1	1020	1020	2	2	2
2	1020	1020	1	1	1
3	1020	1020	1	1	1
4	1020	1020	2	2	2
5	1020	1020	2	2	2
6	1020	1020	1	1	0
7	1020	1020	2	2	2
8	1020	1020	1	1	0
9	1020	1020	1	1	1
10	1020	1020	1	0	0
11	1022	1022	1	1	0
12	1022	1022	1	1	1
13	1025	1024	1	1	1
14	1030	1028	1	1	1
15	1030	1028	1	1	1
16	1035	1032	2	2	2
17	1035	1032	2	2	2
18	1040	1036	1	1	1
19	1045	1040	1	1	1
20	1045	1040	1	1	1
21	1045	1040	1	1	1
22	1049	1043	2	2	2
23	1049	1043	1	1	0
24	1049	1043	1	0	0
25	1050	1044	1	1	1
26	1050	1044	1	1	1
27	1050	1044	2	2	2
28	1050	1044	1	1	1
29	1050	1044	1	1	1
30	1050	1044	1	1	1
31	1055	1048	2	2	2
32	1055	1048	2	2	2
33	1055	1048	2	2	2
34	1055	1048	1	1	1
35	1055	1048	1	0	0
36	1055	1048	1	1	0

Table 25 Continued

#	Schedule time (minutes)	Modified scheduled time	Aircraft type		
			2 types case	3 types 1st case	3 types 2nd case
37	1060	1052	2	2	2
38	1060	1052	2	2	2
39	1070	1060	1	1	0
40	1070	1060	2	2	2
41	1075	1064	2	2	2
42	1075	1064	1	0	0
43	1075	1064	1	1	1
44	1075	1064	2	2	2
45	1080	1068	1	1	1
46	1080	1068	1	1	1
47	1080	1068	1	1	1
48	1080	1068	1	1	1
49	1081	1069	1	1	1
50	1084	1071	1	1	1
51	1085	1072	1	0	0
52	1085	1072	2	2	2
53	1085	1072	2	2	2
54	1085	1072	2	2	2
55	1089	1075	1	1	1
56	1090	1076	2	2	2
57	1090	1076	2	2	2
58	1095	1080	2	2	2
59	1095	1080	1	1	0
60	1095	1080	2	2	2
61	1095	1080	2	2	2
62	1098	1082	2	2	2
63	1100	1084	1	0	0
64	1100	1084	2	2	2
65	1105	1088	1	1	1
66	1105	1088	1	1	0
67	1105	1088	2	2	2
68	1110	1092	1	1	1
69	1110	1092	2	2	2
70	1110	1092	1	1	1
71	1110	1092	2	2	2
72	1110	1092	2	2	2

Table 25 Continued

#	Schedule time (minutes)	Modified scheduled time	Aircraft type		
			2 types case	3 types 1st case	3 types 2nd case
73	1110	1092	1	1	1
74	1110	1092	1	1	1
75	1110	1092	1	1	0
76	1115	1096	2	2	2
77	1115	1096	1	1	1
78	1116	1097	1	0	0
79	1120	1100	2	2	2
80	1120	1100	1	0	0
81	1126	1105	1	1	0
82	1130	1108	1	0	0
83	1135	1112	1	1	0
84	1137	1114	1	1	1
85	1139	1115	1	1	1
86	1139	1115	1	1	1
87	1140	1116	2	2	2
88	1140	1116	2	2	2
89	1140	1116	1	1	1
90	1140	1116	2	2	2
91	1140	1116	2	2	2
92	1140	1116	1	1	1
93	1140	1116	1	1	1
94	1145	1120	2	2	2
95	1145	1120	1	1	0
96	1147	1122	1	1	1
97	1150	1124	1	1	1
98	1150	1124	2	2	2
99	1150	1124	1	1	1
100	1150	1124	2	2	2
Total of type 0			0	9	21
Total of type 1			61	52	40
Total of type 2			39	39	39

Appendix D: Data from Atlanta Airport for the 2 Runways – 2 Types Experiment

Table 26: ATL Airport Data

#	Schedule time (minutes)	Type	Type (3:1 ratio)	Type (1:1 ratio)
1	490	1	2	2
2	490	1	2	2
3	490	1	1	2
4	490	1	2	2
5	490	1	2	2
6	490	2	1	2
7	492	1	1	1
8	492	1	1	2
9	492	1	1	2
10	492	1	1	1
11	493	1	1	2
12	493	1	1	1
13	494	1	1	1
14	494	1	2	2
15	495	1	2	2
16	495	1	1	1
17	495	1	2	2
18	495	1	1	1
19	495	1	1	1
20	495	1	1	1
21	495	1	1	1
22	495	1	1	1
23	496	1	1	2
24	496	1	2	2
25	497	1	1	2
26	497	1	1	1
27	497	1	1	1
28	499	1	2	2
29	500	1	1	1
30	500	1	2	2
31	500	1	1	2
32	500	1	1	2
33	500	1	1	1
34	500	1	1	2
35	501	1	1	1
36	502	1	1	2
52	510	1	1	2
53	511	1	1	2
54	512	1	2	2
55	514	1	1	1
56	515	1	1	1
57	515	1	2	2
58	515	1	1	1
59	515	1	2	2
60	517	1	1	1
61	519	1	1	1
62	520	1	1	1
63	521	1	1	1
64	522	1	1	1
65	524	1	1	1
66	525	1	2	2
67	525	2	2	2
68	525	1	1	1
69	527	1	2	2
70	529	1	1	2
71	529	1	2	2
72	530	1	1	1
73	535	1	1	1
74	535	1	1	2
75	535	1	2	2
76	535	1	1	2
77	535	1	1	2
78	535	1	1	1
79	535	1	1	2
80	535	1	2	2
81	535	1	1	2
82	535	1	1	2
83	535	1	1	1
84	536	1	2	2
85	536	1	2	2
86	537	1	2	2
87	537	1	1	1

Table 26 Continued

#	Schedule time (minutes)	Type	Type (3:1 ratio)	Type (1:1 ratio)
37	502	1	1	1
38	504	1	1	2
39	504	1	2	2
40	504	1	1	1
41	505	1	1	1
42	505	1	1	2
43	505	1	1	1
44	507	1	2	2
45	509	1	1	2
46	509	1	1	1
47	510	1	1	1
48	510	1	1	1
49	510	1	1	2
50	510	1	1	1
51	510	1	1	1

#	Schedule time (minutes)	Type	Type (3:1 ratio)	Type (1:1 ratio)
88	538	1	1	1
89	539	1	1	1
90	540	1	1	2
91	540	1	1	1
92	540	1	1	1
93	540	1	1	2
94	540	1	1	1
95	541	1	2	2
96	541	1	2	2
97	542	1	1	2
98	542	1	1	1
99	543	1	2	2
100	545	1	1	1
Type 1 total		98	73	46
Type 2 total		2	27	54

Appendix E: MATLAB Code to find C_{max} and F_{max} for 1 RW 2 Types Problem

Code E1: Using Small Instances Approach to Solve Single Runway, Two Types Problem

```
%% Input problem parameters
% This code solve approximately the N/1/STsd,rj/Fmax scheduling problem
for special case:
% When there are ONLY 2 types of jobs which implies two common setup times
% A special heuristic (PATH) is used to search within group configuration.
N = 10;     % N >= 3
p = 2;
ST12 = 0;
ST21 = 1;
%r = zeros(1,N); rij = round(random('Uniform',0,2*p,1,N-1)+.5);
%    for i = 2:N, r(i) = r(i-1) + rij(i-1); end
r = [0 1 2 4 5 6 8 10 11 12];
M = [2 2 1 1 2 2 1 1 2 1];
%M = round(random('Uniform',0,1,1,N))+1;
%M = ones(1,N)+1; M(1:round(N/2)) = 1;                 % [1 1 ... 2 2 ...]
%M = ones(1,N); M(1,3:4:end) = 2; M(1,4:4:end) = 2;  % [1 1 2 2 1 1 2
2...]
%M = ones(1,N); M(1,2:2:end) = 2;                     % [1 2 1 2 ...]
%load('Seed_for_M.mat')
%load('Seed_for_r.mat')
%r = rr(1,1:N);
%M = MM(5,1:N);  % seed 5 is used because 14 changes (good for testing)
idxM1 = find(M == 1);
idxM2 = find(M == 2);
M1 = size(idxM1,2);     % M1= No of jobs of type 1
M2 = size(idxM2,2);     % M2= No of jobs of type 2
%% Identifying the number of possible groups and solutions
C = 0;      % C = counter for total number of solutions
if M1 == M2
    N1 = N/2 - 1; N2 = N1;
    idxN1 = idxM1; idxN2 = idxM2;
    G = 2*(N-1);    % G = No of possible groups
    for i = 1: N2
        C = C + (factorial(N2)/(factorial(i-1)*factorial(N2-i+1)))^2 +
(factorial(N2)/(factorial(i-1)*factorial(N2-i+1))) *
(factorial(N2)/(factorial(i)*factorial(N2-i)));
    end
    C = 2*(C + 1);
else % M1 ~= M2
    N1 = max([M1 M2])-1; N2 = min([M1 M2])-1;
    if N1+1 == M1, idxN1=idxM1 ; idxN2=idxM2; else idxN1= idxM2;
idxN2=idxM1; end
    G = 4*(N2+1)-1;     % G = No of possible groups
    for i = 1:N2+1
        C = C + 2*(factorial(N1)/(factorial(i-1)*factorial(N1-i+1))) *
(factorial(N2)/(factorial(i-1)*factorial(N2-i+1))) +
(factorial(N1)/(factorial(i)*factorial(N1-i))) *
(factorial(N2)/(factorial(i-1)*factorial(N2-i+1)));
            if (i < N2+1) && (N2 > 0)
                C = C+(factorial(N1)/(factorial(i-1)*factorial(N1-
i+1)))*(factorial(N2)/(factorial(i)*factorial(N2-i)));
            end
```

```
    end
end
%% Forming and evaluating candidate solutions
CGminC = [1:G; zeros(4,G)]'; CGminC(:,3) = inf; % [idx, no_of_grps,
min_Fmax, no_of_solutions, sol_idx]
CG2minC = zeros(round(size(CGminC,1)/2),4); CG2 = inf; w2=0;
if ST12 == ST21   % IMPORTANT: Gord is not useful if S12 ~= S21 !!!
    Gord = 1; for i = 2:N; if M(i) ~= M(i-1), Gord = Gord+1; end, end
% first criteria to reduce CPU time
end
% IMPORTANT: assumption is jobs already ordered by release time
S = zeros(C,N); F = zeros(C,N); Cmax= zeros(C,1);  % S: index for
schedules F: index for F values in each schedule
GX = 1;     % Groups counter
cX = 1;     % Solutions counter
terminate1 = 0;  % stoping variable
x0 = 1; x1 = 0; x2 = 0; x3 = 0; % control variables
while terminate1 == 0 && GX <= G
    g1 = x0 + x1;     % No of groups of type 1 (more or equal jobs than
type 2)
    g2 = x0 + x2;     % No of groups of type 2
    CGminC(GX,2) = g1 + g2; CGminC(GX,4) = cX;
    if g1>1, cmb1 = combnk(1:N1,g1-1); else cmb1=N1+1; end
    if g2>1, cmb2 = combnk(1:N2,g2-1); else cmb2=N2+1; end
    cmb1 = [zeros(size(cmb1,1),1) cmb1 repmat(N1+1,size(cmb1,1),1)];
    cmb2 = [zeros(size(cmb2,1),1) cmb2 repmat(N2+1,size(cmb2,1),1)];
    for i = 1:size(cmb1,1)
        for j = 1:size(cmb2,1)
            tempg1 = zeros(1,g1); tempg2 = zeros(1,g2);
            for k = 1:g1, tempg1(1,k) = cmb1(i,k+1)-cmb1(i,k); end
            for k = 1:g2, tempg2(1,k) = cmb2(j,k+1)-cmb2(j,k); end
            tempidx1 = idxN1; tempidx2 = idxN2;
            if x3 == 0
                S1 = tempidx1(1:tempg1(1,1));
                tempidx1(1:tempg1(1,1))=[]; tempg1(1)=[];
                xx = 0;
            elseif x3 == 1
                S2 = tempidx2(1:tempg2(1,1));
                tempidx2(1:tempg2(1,1))=[]; tempg2(1)=[];
                xx = 1;
            end
            for jj = 2:g1+g2
                if xx == 0
                    if x3 == 0, S1 = [S1 tempidx2(1:tempg2(1,1))];
tempidx2(1:tempg2(1,1))=[]; tempg2(1)=[];
                    elseif x3 == 1, S2 = [S2 tempidx2(1:tempg2(1,1))];
tempidx2(1:tempg2(1,1))=[];tempg2(1)=[]; end
                    xx = 1;
                elseif xx == 1
                    if x3 == 0, S1 = [S1 tempidx1(1:tempg1(1,1))];
tempidx1(1:tempg1(1,1))=[]; tempg1(1)=[];
                    elseif x3 == 1, S2 =[S2 tempidx1(1:tempg1(1,1))];
tempidx1(1:tempg1(1,1))=[]; tempg1(1)=[]; end
                    xx = 0;
                end
            end
```

```
            if x3 == 0, S(cX,:) = S1; elseif x3 == 1, S(cX,:) = S2; end
        % Evaluating the current solution
            rj(1,1) = r(S(cX,1)); Sij(1,1) = 0; tj(1,1) = rj(1,1); cj(1,1)
= tj(1,1)+ p;
            F(cX,1) = cj(1,1) - rj(1,1);
            for k = 2:N
                rj(1,k) = r(S(cX,k));
                if M(S(cX,k)) == M(S(cX,k-1)), Sij(1,k) = 0;
                elseif M(S(cX,k))==1 && M(S(cX,k-1))==2; Sij(1,k) = ST21;
                elseif M(S(cX,k))==2 && M(S(cX,k-1))==1; Sij(1,k) = ST12;
                end
            tj(1,k) = max([rj(1,k) cj(1,k-1)+Sij(1,k)]);
            cj(1,k) = tj(1,k) + p;
            F(cX,k) = cj(1,k) - rj(1,k);
            end
            Cmax(cX,1)=cj(1,end);
            if Cmax(cX,1) < CGminC(GX,3), CGminC(GX,3) = Cmax(cX,1);
CGminC(GX,5) = cX; end
            cX = cX+1;
        end
    end
    CGminC(GX,4) = cX - CGminC(GX,4);
    if mod(GX,2)==0, CG2minC(GX/2,1:3) = [CGminC(GX,2)
min(CGminC(GX,3),CGminC(GX-1,3)) CGminC(GX,4)+CGminC(GX-1,4)];
        if CG2minC(GX/2,2) <= CG2, CG2 = CG2minC(GX/2,2); w2=0; else w2 =
w2+1; end % second stopping criteria
        CG2minC(GX/2,4)=w2;
    elseif M1~=M2 && GX==G, CG2minC(round(GX/2),1:3) = CGminC(GX,2:4);
        if CG2minC(round(GX/2),2) <= CG2, CG2 = CG2minC(round(GX/2),2);
w2=0; else w2 = w2+1; end % second stopping criteria
        CG2minC(round(GX/2),4)=w2;
    end
    if x1 == 0 && x2 == 0 && x3 == 0, x3 = 1;
    elseif x1 == 0 && x2 == 0 && x3 == 1, x1 = 1; x3 = 0;
    elseif x1 == 1, x1 = 0; x2 = 1; x3 = 1;
    elseif x2 == 1, x0 = x0 + 1; x2 = 0; x3 = 0;
    end
    %if yc==2, terminate = 1; end  % second stopping criteria if no
improvement after two increasing groups
    if ST12==ST21 && 2*x0+x1+x2 > Gord, terminate1 = 1; end        % if
Gord reached (ie g1+g2>Gord) then stop
    GX = GX + 1;
end
if terminate1 == 1, CGminC(GX:end,:)=[]; CG2minC(round(GX/2):end,:)=[];
S(cX:end,:)=[]; F(cX:end,:)=[]; end
plot(CG2minC(:,1),CG2minC(:,2))
[CGminC max(F(CGminC(:,5),:),[],2)]  % [Iteration# NO.Groups Opt.Cmax
NO.Sol]
```

Appendix F: MATLAB Code to Perform Experiments for 1 Runway – 3 Types

Code F1: Using FCFS

```
load('JFK_data_2.mat')
N = size(JFK_rm,1);
S = JFK_rm(:,1)';
M = JFK_rm(:,3)';
r = JFK_rm(:,2)';
SS = S_3types (S,r,M,0,0);
FF = SS(5,:)-SS(1,:);
Cmax = SS(5,end)-r(1)
Usage = utilization_3(SS)
Fmax = max(FF)
Favg = sum(FF)/size(FF,2)
```

Code F2: S_3types Function

```
function [ REPORT ] = S_3types (S,r,M,M0,c0)
%   S_3types summary:
%   This functioin convert a sequence input (with 3 types) to detailed
%   report as in Cmax_function
%   Input (S,r,M,M0,c0)
%   Output S_report (5xN: release time, type, idx, start, completion)
p = 1; ST21 = 1; ST10 = 1; ST20 = 2;
REPORT = zeros(5,size(S,2));
REPORT(1:3,1) = [r(S(1)); M(S(1)); S(1)];
if M(S(1))==1 && M0==2; Sij(1,1) = ST21;
elseif M(S(1))==0 && M0==2; Sij(1,1) = ST20;
elseif M(S(1))==0 && M0==1; Sij(1,1) = ST10;
else
    Sij(1,1) = 0;
end
REPORT(4,1) = max([REPORT(1,1) c0+Sij(1,1)]); %starting time for job 1
REPORT(5,1) = REPORT(4,1)+ p;
for k = 2:size(S,2)
    REPORT(1:3,k) = [r(S(k)); M(S(k)); S(k)];
    if M(S(k))==1 && M(S(k-1))==2; Sij(1,k) = ST21;
    elseif M(S(k))==0 && M(S(k-1))==2; Sij(1,k) = ST20;
    elseif M(S(k))==0 && M(S(k-1))==1; Sij(1,k) = ST10;
    else
        Sij(1,k) = 0;
    end
    REPORT(4,k) = max([REPORT(1,k) REPORT(5,k-1)+Sij(1,k)]);
    REPORT(5,k) = REPORT(4,k)+ p;
end
end
```

Code F3: Using S_3types Function to Perform Improved Approach (Minimize Maximum Delay and Usage)

```
%% Input problem parameters
% This code solve N/1/3-types,rj/Fmax scheduling problem for special case:
% When there are 3 job types which implies 6 common setup times
%=========================================================================
% Initial conditions:
load('JFK_data2.mat')
cycle = 10;    %(Fixed cycle size)
I0 = [];       % Jobs considered from previous cycle
I1 = 1;        % idx for first job in order
I2 = I1-1 + cycle -size(I0,1) ; % last job idx for current search
I3 = 0;             % initial value for M0
I4 = JFK_rm(I1,2); % initial value for c0
z = 1;              % idx for SS table
SS = zeros(5,size(JFK_rm,1)); % Building final schedule based on S_Report
done = 0; iteration = 0;
while done == 0
%% Setting r (release time) and M (type) for jobs in current cycle
iteration = iteration +1;
if I2 >= size(JFK_rm,1)-1, I2 = size(JFK_rm,1); done = 1; end %STOPPING
CONDITION
JFK_rm1 = JFK_rm([I0 I1:I2],:);
[idxS, idxSS] = sort(JFK_rm1(:,1));
N = size(JFK_rm1,1);      % N >= 2
M = JFK_rm1(idxSS,5)';
r = JFK_rm1(idxSS,2)';
%% Forming and evaluating all solutions in current cycle
idxM0 = find(M == 0);
idxM1 = find(M == 1);
idxM2 = find(M == 2);
M0 = size(idxM0,2);      % M0= No of jobs of type 0
M1 = size(idxM1,2);      % M1= No of jobs of type 1
M2 = size(idxM2,2);      % M2= No of jobs of type 2
if M0==N || M1==N || M2==N
    Sopt = S_3types (1:N,r,M,I3,I4);
    Sopt(3,:) = idxS(Sopt(3,:));
    F = Sopt(5,:)-Sopt(1,:);
    Fmax = max(F);
    Favg = sum(F)/size(F,2);
else
A = combnk(1:N,M1);
B = combnk(1:N-M1,M2);
S = zeros(size(A,1)*size(B,1),N);
report = zeros(size(A,1)*size(B,1),3);
Fmax = inf;
k = 1;
for i = 1:size(A,1)
    for j = 1:size(B,1)
        temp = 1:N;
        S(k,A(i,:)) = idxM1;     % assign type 1 to sequence k
        temp(A(i,:)) = [];
        S(k,temp(B(j,:))) = idxM2;  % assign type 2 to sequence k
        temp(B(j,:)) = [];
        S(k,temp) = idxM0;  % assign type 0 to sequence k
```

```
        S1 = S_3types (S(k,:),r,M,I3,I4); % Evaluating current solution
        F = S1(5,:)-S1(1,:);
        report(k,:) = [max(F) sum(F)/size(F,2) S1(5,N)];
        if max(F) < Fmax
            Sopt = S1;
            Sopt(3,:) = idxS(Sopt(3,:));
            Fmax = max(F);
            Favg = sum(F)/size(F,2);
            %report(k,:) %Temperory
        elseif max(F) == Fmax
            if sum(F)/size(F,2) < Favg
                Sopt = S1;
                Sopt(3,:) = idxS(Sopt(3,:));
                Fmax = max(F);
                Favg = sum(F)/size(F,2);
            end
        end
        k = k+1;
    end
end
end
AAA(iteration,:) = [Fmax Favg];  % Testing results
SS(:,z:z+N-1) = Sopt; % EXPORT to Final schedule
%% Next cycle preparation (BASED ON ONE OPTION CRITERIA)
if done == 0
    cc = SS(5,z:z+N-1); %completion time for each job in current cycle
    rr = JFK_rm(I2+1,2);
    I3 = SS(2,z);     %initial M0 for next cycle
    I4 = cc(1);       %initial c0 for next cycle
    I0 = SS(3,z+1:z+N-1); %initial jobs to consider in next cycle
    z = z + 1;
    for i = 2:N
        if cc(i)< rr
            I3 = SS(2,z);   %initial M0 for next cycle
            I4 = cc(i);       %initial c0 for next cycle
            I0(1)=[];     %updated initial jobs list to carry on
            z = z + 1;        %idx for first row to update in SS (ie z-1:
#of fixed rows in SS)
        end
    end
    I1 = I2+1; % first job to consider from the new cycle
    I2 = I1-1 + cycle - size(I0,2);
else done
end
end
%% REPORTS and ANALYSIS
FF = SS(5,:)-SS(1,:);
AAA = AAA(1:iteration,:)
Fmax = max(FF)
Favg = sum(FF)/size(FF,2)
Cmax = SS(5,end)-SS(1,1)
Usage = utilization_3(SS)
```

Appendix G: MATLAB Code to Perform Experiments for 2 Runways – 2 Types

Code G1: Using FCFS

```
%load('TWO_RW_data1.mat')  % Generated data for 100 jobs
load('ATL_data50.mat')  % Collected data from ATL (8:00am-11:00am)
I3 = 1; I4 = 0; %Initial conditions
rL(1:N) = r_gate(1:N) ;%+ terminal(T(1:N),1)'; %Adjusted release times for
RW(L)
rR(1:N) = r_gate(1:N) ;%+ terminal(T(1:N),2)'; %Adjusted release times for
RW(R)
    % RW assignment
S1 = zeros(1,N); S2 = zeros(1,N);
for i = 1:N     % Assign odd numbers to L, even numbers to R
    if mod(i,2)==0  %T(i) == 1 || T(i) == 2
        S2(i) = RW2_ATL(i,1);
    else
        S1(i) = RW2_ATL(i,1);
    end
end
S1(S1==0)=[]; S2(S2==0)=[];
[S1_ORD,idx1] = sort(rL(S1)); S1 = S1(idx1); % Sort jobs based on RW
arrival time
[S2_ORD,idx2] = sort(rR(S2)); S2 = S2(idx2); % Sort jobs based on RW
arrival time
% Sequence evaluation
SL = S_REPORT_function(S1,rL,M,I3,I4)
SR = S_REPORT_function(S2,rR,M,I3,I4)
    % ANALYSIS REPORT
FL = SL(5,:)-SL(1,:);
FR = SR(5,:)-SR(1,:);
minFmaxL = max(FL)
minFmaxR = max(FR)
FLavg = sum(FL)/size(FL,2)
FRavg = sum(FR)/size(FR,2)
minCmaxL = SL(5,size(SL,2))
minCmaxR = SR(5,size(SR,2))
usageL = utilization(SL)
usageR = utilization(SR)
```

Code G2: Using C_{max} Constructive Function to Perform Method A (Focus in Usage)

```
%% Introducing the assignment and the adjusted release times
%load('TWO_RW_data1.mat')  % Generated data for 100 jobs
load('ATL_data1.mat')  % Collected data from ATL (8:00am-11:00am)
%N = 100; % To fit with 1RW code: N >= 8 and each runway: Ni>=3
% Given (r,M,T) and terminals-to-runways taxi-times, find (rL and rR).
%terminal = [0,0;0,0;0,0]; %distance from Terminal to RW
rL(1:N) = r_gate(1:N) ;%+ terminal(T(1:N),1)'; %Adjusted release times for
RW(L)
rR(1:N) = r_gate(1:N) ;%+ terminal(T(1:N),2)'; %Adjusted release times for
RW(R)
%% Considering a cycle (Max 14 job/ cycle for CPU efficiency- as of
6/21/2017)
% (Initial conditions for first cycle)
SSL = zeros(5,N); % Final sequence report for left RW
SSR = zeros(5,N); % Final sequence report for right RW
ii = 1; %counter for the while loop and idx for SS
cycle = 14;     % Minimum is 4
I1 = 1; I2 = cycle;
INPUT = I1:I2;
I3L = 0; I4L = 0; % Initial conditions for left RW
I3R = 0; I4R = 0; % Initial conditions for right RW
%% Find all possible assignments and solve each one as two independent RWs
while ii > 0
Cmax = inf; % initial solution for Cmax*
if size(INPUT,2) >= 6, ASSIGNMENT = floor(size(INPUT,2)/2)-
1:round(size(INPUT,2)/2)+1;
else
    ASSIGNMENT = floor(size(INPUT,2)/2):round(size(INPUT,2)/2);
end
for i= ASSIGNMENT %Only look for combinations around N/2 +-1 (Optional)
    L = combnk(INPUT,i);    %List of all possible assignments for RW(L)
for i jobs
    R = zeros(size(L,1),size(INPUT,2)-size(L,2));
    for j = 1:size(L,1)
        A=INPUT;
        for k = 1:size(L,2)
            A(A==L(j,k))= [];
        end
        R(j,:)=A; %List of all possible assignments for RW(R) for
remaining jobs
    end
    for j = 1:size(L,1)
        % Getting required r and M for each runway
        r1 = rL(L(j,:));
        r2 = rR(R(j,:));
        M1 = M(L(j,:));
        M2 = M(R(j,:));
    % GOTO single RW function. Solve the 2 assigned jobs in row j as
two...
    % ...independent single RWs.
        [CmaxL,FmaxL,SL,M00L,c00L] =
Cmax_construct_function(size(L,2),r1,M1,I3L,I4L);
        [CmaxR,FmaxR,SR,M00R,c00R] =
Cmax_construct_function(size(R,2),r2,M2,I3R,I4R);
```

```
        %UL = utilization(SL);
        %UR = utilization(SR);
        %TCmax(j)= CmaxL+CmaxR;
        % compare the results and return
        if max(CmaxL,CmaxR) < Cmax
            Cmax = max(CmaxL,CmaxR);
            usage = CmaxL+CmaxR;
            SLopt = SL; SRopt = SR;
            SLopt(3,:)= L(j,SL(3,:)); % Using original job index of the
main problem
            SRopt(3,:)= R(j,SR(3,:)); % Using original job index of the
main problem
        elseif max(CmaxL,CmaxR) == Cmax     % Tie Breaker
            if CmaxL+CmaxR < usage
                Cmax = max(CmaxL,CmaxR);
                usage = CmaxL+CmaxR;
                SLopt = SL; SRopt = SR;
                SLopt(3,:)= L(j,SL(3,:)); % Using original job index of
the main problem
                SRopt(3,:)= R(j,SR(3,:)); % Using original job index of
the main problem
            end
        end
    %(SUGGESTION: Need to pick a sequence with least changes from the
previous
    %cycles sequence (to minimize the number of posible changes in
sequences))
    end
end
%% Solution report (How to record final solution- Recording gradually)
% (Take the first two jobs assigned in each RW sequence and report them)
SSL(:,ii)= SLopt(:,1);
SSR(:,ii)= SRopt(:,1);
% Next cycle preparation (How to make it work automaticly?)
% Consider adding two new jobs to the mix replacing first two jobs and
doing necessary adjustments
I3L = SLopt(2,1); I4L = SLopt(5,1);   % Initial conditions for next cycle
I3R = SRopt(2,1); I4R = SRopt(5,1);   % Initial conditions for next cycle
if size(SLopt,2)>=2 && size(SRopt,2)>=2  % Carrying rest of jobs other
than first two
    I0 = [SLopt(3,2:end), SRopt(3,2:end)];
elseif size(SLopt,2)>=2 && size(SRopt,2)<2      % This might not be
needed!
    I0 = SLopt(3,2:end);
elseif size(SLopt,2)<2 && size(SRopt,2)>=2      % This might not be
needed!
    I0 = SRopt(3,2:end);
end
I1 = I2+1; I2 = I2+2;   % Considering the next two available jobs
if I1 <= N
    INPUT1 = [I0, I1];
    if I2 <= N
        INPUT1 = [I0, I1, I2];
    end
    INPUT = sort(INPUT1);
    ii = ii + 1;
```

```
else
    SSL(:,ii:ii+size(SLopt,2)-1)= SLopt(:,:);
    SSR(:,ii:ii+size(SRopt,2)-1)= SRopt(:,:);
    ii = 0;
end
% TESTING AREA
INPUT
%if ii>0 && ii<7, SSL(3,1:10), SSR(3,1:10), INPUT, %pause,
%elseif ii>=7, SSL(3,ii-5:ii+4), SSR(3,ii-5:ii+4), INPUT, %pause,
%end
end % while loop
% TESTING AREA (ANALYSIS)
SeqLF = SSL(3,:); SeqLF(SeqLF==0) = [];
SeqRF = SSR(3,:); SeqRF(SeqRF==0) = [];
SSL(:,1:size(SeqLF,2))
SSR(:,1:size(SeqRF,2))
FL = SSL(5,1:size(SeqLF,2))-SSL(1,1:size(SeqLF,2))
FR = SSR(5,1:size(SeqRF,2))-SSR(1,1:size(SeqRF,2))
minFmaxL = max(FL)
minFmaxR = max(FR)
FLavg = sum(FL)/size(FL,2)
FRavg = sum(FR)/size(FR,2)
minCmaxL = SSL(5,size(SeqLF,2))
minCmaxR = SSR(5,size(SeqRF,2))
utilization(SSL(:,1:size(SeqLF,2)))
utilization(SSR(:,1:size(SeqRF,2)))
```

Code G2a: C_{max} Constructive Function

```
function [Cmax,Fmax,S,M00,c00] = Cmax_construct_function(N,r,M,M0,c0)
%Cmax_construct_function
%   This function solve the N/1/Pj=P,STsd,rj/Cmax RW scheduling problem
for special case:
%   When there are ONLY 2 types of jobs which implies TWO common setup
times.
%   A constructive approach is used to have a gaurenteed optimal solution
for Cmax
%   =>Input: {N,r,M,M0,c0}
%       N: Numner of jobs
%       r: Release times
%       M: Job type {1,2}
%       M0: Initial type (0 if no initial type, 1 if type 1, 2 if type 2)
%       c0: intial completion time (0 if no initial completion time)
%   <=Output: {Cmax,Fmax,S,M00,c00}
%       Cmax
%       Fmax
%       S:(release time, type, idx, start time, completion time)
%       M00: initital type to carry for next cycle
%       c00: inital completion time to carry for next cycle

%================================START==============================
% Given N, r, M, M0, c0 from main code
p = 1; ST12 = 0; ST21 = 1; %NOTE: the code is built such that S12=0 AND
S21=1 to work correctly
                           % it can be modified, if needed. (4/28/17)
ORM = zeros(3,N); % ordered by release time input (r,M)
[ORM(1,:) ,idxORM] = sort(r,2); ORM(2,:) = M(idxORM); ORM(3,:) = idxORM;

%====================================================================
% Forming and evaluating optimal solution
    % part 1 - Sequencing first job
S = zeros(5,N);             % S:(release time, type, idx, start time,
completion time)
c1 = 1; % counter for searching
c4 = 0; % 0-1 switch to delete the dummy job at the end
if M0 == 0
    if ORM(1,1) < ORM (1,2)     % if only one job available at the
beginning
        S(1:3,1) = ORM(:,1);    % then, it is sequenced first
        S(4,1) = ORM(1,1);
        S(5,1) = ORM(1,1)+p;
        ORM(:,1)=[];
    else                        % but if more than one job is available
        while c1 ~= 0           % then, pick the first type 1 job
            if ORM(1,c1) == ORM(1,1)
                if ORM(2,c1) == 1
                    S(1:3,1) = ORM(:,c1); S(4,1) = ORM(1,c1); S(5,1) =
ORM(1,c1)+p;
                    ORM(:,c1)=[]; c1=0;
                else
                    if c1 == size(ORM,2)
                        S(1:3,1) = ORM(:,1); S(4,1) = ORM(1,1); S(5,1) =
ORM(1,1)+p;
```

```
                    ORM(:,1)=[]; c1=0;
                else
                    c1 = c1+1;
                end
            end
        else    % but if no type 1 job available, then, pick first
type 2 job
            S(1:3,1) = ORM(:,1); S(4,1) = ORM(1,1); S(5,1) =
ORM(1,1)+p;
            ORM(:,1)=[]; c1=0;
        end
    end
end
    S = zeros(5,N+1);     % S:(release time, type, idx, start time,
completion time)
    S(:,1) = [0; M0; 0; c0-p; c0]; % dummy job to be taken out from S
after sequincing all jobs
    c4 = 1; % turn switch on to delete dummy job at the end
end
    % part 2 - Sequencing the rest of the jobs
c3 = 2; % c3: index for columns in S
while size(ORM,2) >= 1
    c1 = 1; c2 = 1; X1 = zeros(4,1); X1(1,1)=inf; X2 = zeros(4,1);
X2(1,1)=inf;
    while c1 ~= size(ORM,2)+1   % locate the first type 1 job (if
available)
        if ORM(2,c1) == 1
            X1(1:3,1) = ORM(:,c1); X1(4,1)=c1; c1=size(ORM,2)+1;
        else
            c1 = c1+1;
        end
    end
    while c2 ~= size(ORM,2)+1   % locate the first type 2 job (if
available)
        if ORM(2,c2) == 2
            X2(1:3,1) = ORM(:,c2); X2(4,1)=c2; c2=size(ORM,2)+1;
        else
            c2 = c2+1;
        end
    end
    if X2(2,1) == 0 || (X1(2,1) == S(2,c3-1) && X1(1,1) <= S(5,c3-1))
||...
        (X1(2,1) == S(2,c3-1) && X1(1,1) > S(5,c3-1) && X1(1,1) <=
X2(1,1)) ||...
        (X1(2,1) ~= S(2,c3-1) && X2(1,1) > S(5,c3-1) && X1(1,1) <=
X2(1,1))
                % i.e. if no type 2 available OR Type 1 avaialable before
                % completion of previous type 1(and before the end of
separation time
                % needed for switching to the other type OR before the
release of
                % the candidate type 2) then use X1
        S(1:3,c3) = X1(1:3,1);
        if X1(2,1) == S(2,c3-1)
            S(4,c3) = max(S(5,c3-1),X1(1,1));
```

```
        else
            S(4,c3) = max(S(5,c3-1)+ST21,X1(1,1));
        end
        S(5,c3) = S(4,c3)+ p;
        ORM(:,X1(4,1))=[];
        c3 = c3 + 1;
    elseif X1(2,1) == 0 || (X2(2,1) == S(2,c3-1) && X2(1,1) <= S(5,c3-1))
||...
        (X2(2,1) == S(2,c3-1) && X2(1,1) > S(5,c3-1) && X2(1,1) < X1(1,1))
||...
        (X2(2,1) ~= S(2,c3-1) && X1(1,1) > S(5,c3-1) && X2(1,1) < X1(1,1))
                % i.e. if no type 1 available OR Type 2 avaialable before
                % completion of previous type 2 (and before the end of
separation time
                % needed for switching to the other type OR before the
release of
                % the candidate type 1) then use X2
        S(1:3,c3) = X2(1:3,1);
        if X2(2,1) == S(2,c3-1)
            S(4,c3) = max(S(5,c3-1),X2(1,1));
        else
            S(4,c3) = max(S(5,c3-1)+ST12,X2(1,1));
        end
        S(5,c3) = S(4,c3)+ p;
        ORM(:,X2(4,1))=[];
        c3 = c3 + 1;
    end
end
if c4 == 1
    S(:,1)= [];
end
M00 = S(2,1); c00 = S(5,1); % Outputs (needed when taking first job out
from next cycle)
F = S(5,:)-S(1,:); Fmax = max(F); %Favg = sum(F)/N;
Cmax = S(5,N); % -S(4,1); (If release of first job is considered as
starting point)
end
```

Code G3: Using F_{max} Cut Function to Perform Method B (Focus in Delay)

```
%% Introducing the assignment and the adjusted release times
load('ATL_data25.mat')  % Collected data from ATL (8:00am-11:00am)
%N = 100; % To fit with 1RW code: N >= 8 and each runway: Ni>=3
% Given (r,M,T) and terminals-to-runways taxi-times, find (rL and rR).
rL(1:N) = r_gate(1:N) ;%+ terminal(T(1:N),1)'; %Adjusted release times for
RW(L)
rR(1:N) = r_gate(1:N) ;%+ terminal(T(1:N),2)'; %Adjusted release times for
RW(R)
%% Considering a cycle (Max 12 job/ cycle for CPU efficency- as of
6/29/2017)
% (Initial conditions for first cycle)
SSL = zeros(5,N); % Final sequence report for left RW
SSR = zeros(5,N); % Final sequence report for right RW
ii = 1; %counter for the while loop and idx for SS
cycle = 12;     % Minimum is 4
I1 = 1; I2 = cycle;
INPUT = I1:I2;
I3L = 0; I4L = 0; % Initial conditions for left RW
I3R = 0; I4R = 0; % Initial conditions for right RW
%% Find all possible assignments and solve each one as two independint RWs
while ii > 0
Fmax = inf; % initial solution for Cmax
if size(INPUT,2) >= 6, ASSIGNMENT = floor(size(INPUT,2)/2)-
1:round(size(INPUT,2)/2)+1;
                                        %ANOTHER OPTION: 2:size(INPUT,2)-2;
else ASSIGNMENT = floor(size(INPUT,2)/2):round(size(INPUT,2)/2);
end
for i= ASSIGNMENT %Only look for combinations around N/2 +-1 (Optional)
    L = combnk(INPUT,i);    %List of all possible assignments for RW(L)
for i jobs
    R = zeros(size(L,1),size(INPUT,2)-size(L,2));
    for j = 1:size(L,1)
        A=INPUT;
        for k = 1:size(L,2)
            A(A==L(j,k))= [];
        end
        R(j,:)=A; %List of all possible assignments for RW(R) for
remaining jobs
    end
    for j = 1:size(L,1)
        % Getting required r and M for each runway
        r1 = rL(L(j,:));
        r2 = rR(R(j,:));
        ML = M(L(j,:));
        MR = M(R(j,:));
    % GOTO single RW function. Solve the 2 assigned jobs in row j as
two...
    % ...independent single RWs.
    if sum(ML,2) == size(ML,2) || sum(ML,2) == 2*size(ML,2) % if all jobs
of same type
        [CmaxL,FmaxL,SL,I3LO,I4LO] =
Cmax_construct_function(size(L,2),r1,ML,I3L,I4L);
    else
```

```
        [CmaxL,FmaxL,S,I3LO,I4LO] =
Fmax_cut_function(size(L,2),r1,ML,I3L,I4L);
        SL = S_REPORT_function(S,r1,ML,I3L,I4L);
    end
    if sum(MR,2) == size(MR,2) || sum(MR,2) == 2*size(MR,2) % if all jobs
of same type
        [CmaxR,FmaxR,SR,I3RO,I4RO] =
Cmax_construct_function(size(R,2),r2,MR,I3R,I4R);
    else
        [CmaxR,FmaxR,S,I3RO,I4RO] =
Fmax_cut_function(size(R,2),r2,MR,I3R,I4R);
        SR = S_REPORT_function(S,r2,MR,I3R,I4R);
    end
    % compare the results and return (Picking a candidate)
        if max(FmaxL,FmaxR) < Fmax
            Fmax = max(FmaxL,FmaxR);
            TFmax = FmaxL + FmaxR;
            SLopt = SL; SRopt = SR;
            SLopt(3,:)= L(j,SL(3,:)); % Using original job index of the
main problem
            SRopt(3,:)= R(j,SR(3,:)); % Using original job index of the
main problem
        elseif max(FmaxL,FmaxR) == Fmax     % Tie Breaker
            if FmaxL + FmaxR < TFmax
                TFmax = max(FmaxL,FmaxR);
                SLopt = SL; SRopt = SR;
                SLopt(3,:)= L(j,SL(3,:)); % Using original job index of
the main problem
                SRopt(3,:)= R(j,SR(3,:)); % Using original job index of
the main problem
            end
        end
    end
end
%% Solution report (How to record final solution- Recording gradually)
% (Take the first two jobs assigned in each RW sequence and report them)
SSL(:,ii)= SLopt(:,1);
SSR(:,ii)= SRopt(:,1);
% Next cycle preparation (How to make it work automaticly?)
% Consider adding two new jobs to the mix replacing first two jobs and
doing necessary adjustments
I3L = SLopt(2,1); I4L = SLopt(5,1);   % Initial conditions for next cycle
I3R = SRopt(2,1); I4R = SRopt(5,1);   % Initial conditions for next cycle
%I3L = I3LO; I4L = I4LO; %(In case of using the function's outpot)
%I3R = I3RO; I4R = I4RO; %(In case of using the function's outpot)
if size(SLopt,2)>=2 && size(SRopt,2)>=2  % Carrying rest of jobs other
than first two
    I0 = [SLopt(3,2:end), SRopt(3,2:end)];
elseif size(SLopt,2)>=2 && size(SRopt,2)<2      % This might not be
needed!
    I0 = SLopt(3,2:end);
elseif size(SLopt,2)<2 && size(SRopt,2)>=2      % This might not be
needed!
    I0 = SRopt(3,2:end);
end
I1 = I2+1; I2 = I2+2;   % Considering the next two available jobs
```

```
if I1 <= N
    INPUT1 = [I0, I1];
    if I2 <= N
        INPUT1 = [I0, I1, I2];
    end
    INPUT = sort(INPUT1);
    ii = ii + 1;
else
    SSL(:,ii:ii+size(SLopt,2)-1)= SLopt(:,:);
    SSR(:,ii:ii+size(SRopt,2)-1)= SRopt(:,:);
    ii = 0;
end
% TESTING AREA (PROGRESS ANALYSIS)
INPUT
%if ii>0 && ii<7, SSL(3,1:10), SSR(3,1:10), INPUT, %pause,
%elseif ii>=7, SSL(3,ii-5:ii+4), SSR(3,ii-5:ii+4), INPUT, %pause,
%end
end % while loop
% TESTING AREA (ANALYSIS)
SeqLF = SSL(3,:); SeqLF(SeqLF==0) = [];
SeqRF = SSR(3,:); SeqRF(SeqRF==0) = [];
SSL(:,1:size(SeqLF,2))
SSR(:,1:size(SeqRF,2))
FL = SSL(5,1:size(SeqLF,2))-SSL(1,1:size(SeqLF,2))
FR = SSR(5,1:size(SeqRF,2))-SSR(1,1:size(SeqRF,2))
minFmaxL = max(FL)
minFmaxR = max(FR)
FLavg = sum(FL)/size(FL,2)
FRavg = sum(FR)/size(FR,2)
minCmaxL = SSL(5,size(SeqLF,2))
minCmaxR = SSR(5,size(SeqRF,2))
utilization(SSL(:,1:size(SeqLF,2)))
utilization(SSR(:,1:size(SeqRF,2)))
```

Code G3a: F_{max} Cut Function

```
function [CmaxOPT,FmaxOPT,SOPT,M00,c00] = Fmax_cut_function(N,r,M,M0,c0)
%Fmax_cut_function
%   This function solve the N/1/Pj=P,STsd,rj/Fmax RW scheduling problem
for special case:
%   When there are ONLY 2 types of jobs which implies TWO common setup
times.
%   A search with stopping criteria approach is used to have a gaurenteed
optimal solution for Fmax
%   =>Input: {N,r,M,I3,I4}
%       N: Number of jobs
%       r: Release times
%       M: Job type {1,2}
%       M0: Initial type (1 if type 1 or no initial condition, 2 if type
2)
%       c0: intial completion time (0 if no initial completion time)
%   <=Output: {Cmin, Sopt, I3, I4}
%       Cmax
%       Fmax: optimal value for Fmax (Cmin for tie breaker)
%       Sopt: Associated optimal sequence
%       M00: initital type to carry for next cycle
%       c00: inital completion time to carry for next cycle

%==================================START================================
% Given N, r, M, I3, I4 from main code
p = 1; ST12 = 0; ST21 = 1;
if M0 == 0, M0=1; end % If no input, then assume type 1 (syntax issue)
idxM1 = find(M == 1);
idxM2 = find(M == 2);
M1 = size(idxM1,2);     % M1= No of jobs of type 1
M2 = size(idxM2,2);     % M2= No of jobs of type 2
%% Identifying the number of possible groups and solutions
C = 0;      % C = counter for total number of solutions
if M1 == M2
    N1 = N/2 - 1; N2 = N1;
    idxN1 = idxM1; idxN2 = idxM2;
    G = 2*(N-1);    % G = No of possible groups
    for i = 1: N2
        C = C + (factorial(N2)/(factorial(i-1)*factorial(N2-i+1)))^2 +
(factorial(N2)/(factorial(i-1)*factorial(N2-i+1))) *
(factorial(N2)/(factorial(i)*factorial(N2-i)));
    end
    C = 2*(C + 1);
else % M1 ~= M2
    N1 = max([M1 M2])-1; N2 = min([M1 M2])-1;
    if N1+1 == M1, idxN1=idxM1 ; idxN2=idxM2; else idxN1= idxM2;
idxN2=idxM1; end
    G = 4*(N2+1)-1;     % G = No of possible groups
    for i = 1:N2+1
        C = C + 2*(factorial(N1)/(factorial(i-1)*factorial(N1-i+1))) *
(factorial(N2)/(factorial(i-1)*factorial(N2-i+1))) +
(factorial(N1)/(factorial(i)*factorial(N1-i))) *
(factorial(N2)/(factorial(i-1)*factorial(N2-i+1)));
            if (i < N2+1) && (N2 > 0)
```

```
                C = C+(factorial(N1)/(factorial(i-1)*factorial(N1-
i+1)))*(factorial(N2)/(factorial(i)*factorial(N2-i)));
            end
    end
end
%% Forming and evaluating candidate solutions
S = zeros(C,N); F = zeros(C,N); Cmax = zeros (C,1);
FGmin = [1:G; zeros(6,G)]'; FGmin(:,3) = inf;
      % [1-idx 2-no_of_grps 3-min_Fmax 4-no_of_solutions 5-sol_idx 6-min
Cmax/Fmax* 7-Fmean/Fmax*]
Gord = 1; for i = 2:N; if M(i) ~= M(i-1), Gord = Gord+1; end, end     %
first criteria to reduce CPU time
GX = 1;     % Groups counter
cX = 1;     % Solutions counter
x0 = 1; x1 = 0; x2 = 0; x3 = 0; % control variables
terminate = 0;  % stoping variable
FG2min = zeros(round(size(FGmin,1)/2),4); FG2 = inf; yc=0;
while terminate == 0 && GX <= G
    g1 = x0 + x1;      % No of groups of type 1 (more or equal jobs than
type 2)
    g2 = x0 + x2;      % No of groups of type 2
    FGmin(GX,2) = g1 + g2; FGmin(GX,4) = cX;
    if g1>1, cmb1 = combnk(1:N1,g1-1); else cmb1=N1+1; end
    if g2>1, cmb2 = combnk(1:N2,g2-1); else cmb2=N2+1; end
    cmb1 = [zeros(size(cmb1,1),1) cmb1 repmat(N1+1,size(cmb1,1),1)];
    cmb2 = [zeros(size(cmb2,1),1) cmb2 repmat(N2+1,size(cmb2,1),1)];
    for i = 1:size(cmb1,1)
        for j = 1:size(cmb2,1)
            tempg1 = zeros(1,g1); tempg2 = zeros(1,g2);
            for k = 1:g1, tempg1(1,k) = cmb1(i,k+1)-cmb1(i,k); end
            for k = 1:g2, tempg2(1,k) = cmb2(j,k+1)-cmb2(j,k); end
            tempidx1 = idxN1; tempidx2 = idxN2;
            if x3 == 0
                S1 = tempidx1(1:tempg1(1,1));
                tempidx1(1:tempg1(1,1))=[]; tempg1(1)=[];
                xx = 0;
            elseif x3 == 1
                S2 = tempidx2(1:tempg2(1,1));
                tempidx2(1:tempg2(1,1))=[]; tempg2(1)=[];
                xx = 1;
            end
            for jj = 2:g1+g2
                if xx == 0
                    if x3 == 0, S1 = [S1 tempidx2(1:tempg2(1,1))];
tempidx2(1:tempg2(1,1))=[]; tempg2(1)=[];
                    elseif x3 == 1, S2 = [S2 tempidx2(1:tempg2(1,1))];
tempidx2(1:tempg2(1,1))=[];tempg2(1)=[]; end
                    xx = 1;
                elseif xx == 1
                    if x3 == 0, S1 = [S1 tempidx1(1:tempg1(1,1))];
tempidx1(1:tempg1(1,1))=[]; tempg1(1)=[];
                    elseif x3 == 1, S2 =[S2 tempidx1(1:tempg1(1,1))];
tempidx1(1:tempg1(1,1))=[]; tempg1(1)=[]; end
                    xx = 0;
                end
            end
```

```
            if x3 == 0, S(cX,:) = S1; elseif x3 == 1, S(cX,:) = S2; end
        % Evaluating the current solution
            rj(1,1) = r(S(cX,1));
            if M(S(cX,1)) == M0, Sij(1,1) = 0;
            elseif M(S(cX,1))==1 && M0==2; Sij(1,1) = ST21;
            elseif M(S(cX,1))==2 && M0==1; Sij(1,1) = ST12;
            end
            tj(1,1) = max([rj(1,1) c0+Sij(1,1)]);
            cj(1,1) = tj(1,1)+ p;
            F(cX,1) = cj(1,1) - rj(1,1);
            for k = 2:N
                rj(1,k) = r(S(cX,k));
                if M(S(cX,k)) == M(S(cX,k-1)), Sij(1,k) = 0;
                elseif M(S(cX,k))==1 && M(S(cX,k-1))==2; Sij(1,k) = ST21;
                elseif M(S(cX,k))==2 && M(S(cX,k-1))==1; Sij(1,k) = ST12;
                end
                tj(1,k) = max([rj(1,k) cj(1,k-1)+Sij(1,k)]);
                cj(1,k) = tj(1,k) + p;
                F(cX,k) = cj(1,k) - rj(1,k);
            end
            Cmax(cX) = cj(1,k);
            % Recording results for FGmin
            if max(F(cX,:)) < FGmin(GX,3)
                FGmin(GX,3) = max(F(cX,:)); FGmin(GX,5) = cX;
                FGmin(GX,6)= Cmax(cX); FGmin(GX,7)= mean(F(cX,:),2);
            elseif max(F(cX,:)) == FGmin(GX,3)
                if Cmax(cX) < FGmin(GX,6)
                    FGmin(GX,5) = cX; FGmin(GX,6)= Cmax(cX); FGmin(GX,7)=
mean(F(cX,:),2);
                elseif Cmax(cX)==FGmin(GX,6)
                    if mean(F(cX,:),2)<= FGmin(GX,7)
                        FGmin(GX,5) = cX; FGmin(GX,7)= mean(F(cX,:),2);
                    end
                end
            end
            cX = cX+1;
        end
    end
    FGmin(GX,4) = cX - FGmin(GX,4);
    if mod(GX,2)==0, FG2min(GX/2,1:3) = [FGmin(GX,2)
min(FGmin(GX,3),FGmin(GX-1,3)) FGmin(GX,4)+FGmin(GX-1,4)];
        if FG2min(GX/2,2) <= FG2, FG2 = FG2min(GX/2,2); yc=0; else yc =
yc+1; end % warning!!
        FG2min(GX/2,4)=yc;
    elseif M1~=M2 && GX==G, FG2min(round(GX/2),1:3) = FGmin(GX,2:4);
        if FG2min(round(GX/2),2) <= FG2, FG2 = FG2min(round(GX/2),2);
yc=0; else yc = yc+1; end
        FG2min(round(GX/2),4)=yc;
    end
    if x1 == 0 && x2 == 0 && x3 == 0, x3 = 1;
    elseif x1 == 0 && x2 == 0 && x3 == 1, x1 = 1; x3 = 0;
    elseif x1 == 1, x1 = 0; x2 = 1; x3 = 1;
    elseif x2 == 1, x0 = x0 + 1; x2 = 0; x3 = 0;
    end
    GX = GX + 1;
    if yc==2, terminate = 1; end  % stopping criteria
```

```
end
if terminate == 1,
    FGmin(GX:end,:)=[]; FG2min(round(GX/2):end,:)=[]; S(cX:end,:)=[];
F(cX:end,:)=[];
end
%% OUTPUT
FminCmin = FGmin(1,[2 3 6 7 5]); % Candidate solution [#groups Fmax Cmax
Favg idx]
for i = 2:size(FGmin,1)
    if FGmin(i,3) < FminCmin(2) % Best Fmax first
        FminCmin = FGmin(i,[2 3 6 7 5]);
    elseif FGmin(i,3) == FminCmin(2) % Tie breaker
        if FGmin(i,6) < FminCmin(3)  % best Cmax
            FminCmin = FGmin(i,[2 3 6 7 5]);
        elseif FGmin(i,6) == FminCmin(3)    % Tie breaker
            if FGmin(i,7) < FminCmin(4)     % best Fmean
                FminCmin = FGmin(i,[2 3 6 7 5]);
            end
        end
    end
end
SOPT = S(FminCmin(5),:);
CmaxOPT = FminCmin(3);
FmaxOPT = FminCmin(2);
M00 = M(SOPT(1));
c00 = c0 + p;
end
```