Algorithms of Scheduling Aircraft Landing Problem

Min Wen

Supervisors: Jens Clausen and Jesper Larsen

Master thesis
Department of Informatics and Mathematical Modelling
Technical University of Denmark

November 2005

Table of Contents

List of Tables

List of Figures

Abstract

This thesis addresses the problem of scheduling aircraft landings at an airport. Given a set of planes and runways, the objective is to minimize the total (weighted) deviation from the target landing time for each plane. There are costs associated with landing either earlier or later than a target landing time for each plane. Each plane has to land on one of the runways within its predetermined time windows such that separation criteria between all pairs of planes are satisfied. This type of problem is a large-scale optimization problem, which occurs at busy airports where making optimal use of the bottleneck resource (the runways) is crucial to keep the airport operating smoothly.

This thesis is the first attempt to develop a branch-and-price exact algorithm for the Aircraft Landing Problem (ALP), in which the column generation method is applied to solve the linear relaxation problem for each branch node throughout the branch-and-bound procedure. We formulate the ALP as a set partitioning problem with side constraints on the number of available runways. We also present a mixed integer formulation for the subproblem in column generation, which can be used to generate the columns with the minimum negative reduced cost. Then a branch-and-bound method is developed to find the optimal integer solution for the ALP. Finally, the branch-and-price exact algorithm is implemented and tested on the public data from OR_Library

involving up to 50 aircraft and 4 runways. The computational results show that the algorithm can solve the problem optimally in acceptable CPU time. Furthermore, the optimal solutions can be achieved with less than 500 columns generated and 12 branch nodes explored.

Acknowledgements

I would like to express my sincere acknowledgement to my supervisors Jens Clausen and Jesper Larsen for their suggestions, support and encouragement during this thesis. Without them this work would never have come into existence.

I am thankful to Brian Kallehauge for the fruitful discussions and the good suggestions. I would like to thank Thomas K. Stidsen for his assistance in programming. I am also grateful to all of the support staff at the Department of IMM, for a making a friendly atmosphere that was conducive to research.

Finally, I would like to thank my friends who assisted during this thesis and my family for their patience and *love*.

Lyngby, Nov 2005
Min Wen

Chapter 1

Introduction

1.1 Motivation

Over the past few decades, air traffic has experienced a tremendous growth. As an example, the world's largest airport - Atlanta, USA - alone handles more than 80 million embarkations and disembarkations per year. The biggest cargo airport, located in Memphis, TN, transports 2.5 million tons of cargo each year. Congonhas, the Brazilian airport through which passes the largest number of aircraft, manages an average of 22 thousand movements a month (Mello (2002)). At Sydney airport, landing slots are allocated at 3 minuts intervals (Ciesielski *et al.* (1998)). Air transport has definitely established itself as one of the most important means of transport in the future.

However, as the air traffic develops, the limitation of the runway becomes the bottleneck during the airport operation. For example, London Heathrow airport, one of the busiest airports in the world, has only two runways (Atkin *et al.* (2004)). When the number of approaching flights exceeds the airport capacity, some of these aircraft can not be landed on its $'$perfect$'$ landing time. There is a cost mainly on the waste of fuel for each plane flying faster than its most economical speed. Airlines also experience different costs for delays of different

flights. Depending on the amount of delay, there might be a number of transfer passengers that miss their connecting flight. The crew or aircraft might also be needed to perform a next flight, that now has to be rescheduled. This might propagate delays to departing flights. There are a lot of other possible costs resulting from delays, such as ground crew rescheduling, crew-overtime payments and so on (Soomer *et al.* (2005)). It has been estimated the cost caused by air traffic congestion would be $10 billion in Europe by the year 2000 (Ciesielski *et al.* (1998)). Therefore solving the aircraft landing problem (ALP), which is the problem of assigning each aircraft an optimal landing time and runway such that the total cost is minimized, is an important area of air traffic operations.

If the terms ′aircraft′ and ′runway′ are considered more loosely, many routing and scheduling problems can be regarded as ALP. An example is that we have a number of customers in need to be picked up by a set of vehicles with given time windows for each customer and the travelling time between any pair of them. This routing problem can be viewed as an ALP where the runways represent the vehicles and the aircraft represent the customers. Another example is to assign a number of jobs on a set of machines where the release time, latest finish time and processing time for each job are given. This scheduling problem can also be considered as an ALP. An in-depth description about the ALP is introduced in section 2.1.

The aircraft landing problem is ′hard′ to solve since it can be viewed as a job machine scheduling problem with release times and sequence-dependent processing time. The job machine scheduling problem has been proved to be NP-hard, hence the ALP is NP-hard (see Beasley *et al*. (2000)). In the past decades, both exact algorithms and heuristic algorithms have been developed for the ALP. The exact algorithms can find the optimal solution within reasonable time for the instances involving up to 50 aircraft. However, as the size increases, the running time consumed for finding the optimal solution by the exact algorithm

usually increase exponentially. Hence, many heuristic methods have also been developed with the hope to solve the problem more quickly, such as population heuristic approaches (see Pinol *et al.* (2003)), time segment heuristic (see Jung *et al.* (2003)) etc. A review of the literature is presented in section 2.3.

In this thesis, we focus on investigating and developing new methods for the aircraft landing problem. We are interested in finding optimal solutions for the large-scale ALP. The exact algorithm, presented by Beasley *et al.* (2000), is based on the LP-based tree search method (so called branch-and-bound) to find the optimal integer solution. Rather than using the branch-and-bound method, we instead apply a branch-and-price approach to find optimal solutions, which has been applied successfully to solve large instances of well known NP-hard problems such as the Vehicle Routing Problem (Larsen, (1999)), Aircrew Scheduling Problems (Desaulniers *et al.* (2001)), Job Machine Scheduling Problems (Chen *et al.* (1997)) etc.

In the branch-and-price methods, column generation is applied for solving the linear relaxation problems throughout the branch-and-bound tree. Column generation, first proposed by Dantzig and Wolfe in 1961, is a powerful mehtod for dealing with linear programming problems which contain a huge number of variables. It starts by solving a restricted problem including only a few variables and then checks the optimality by a pricing problem. However, as mentioned in Barnhart *et al.* (1998), there are fundamental difficulties in applying column generation techniques for linear programming in integer programming solution methods, including the following:

- Conventional integer programming branching on variables may not be effective because fixing variables can destroy the structure of the pricing problem.

- Solving the linear programming problems and the subproblems to optimality may not be efficient, in which case different rules will apply for managing the branch-and-price tree.

For a general survey of the branch-and-price method see Barnhart *et al.* (1998).

1.2 Overview of the contribution of the thesis

This thesis is the first attempt to develop a branch-and-price based algorithm for the ALP. Contributions are made on the following aspects: We present a set partitioning formulation for the problem of which the linear relaxation provides excellent lower bound for the integer problem. We propose a mixed integer formulation for solving the subproblem in the column generation. The model not only determines the column with the minimum reduced cost that is to be added to the master problem, but also solves the sequence problem and gives the landing time for the aircraft appearing in the column. Moreover, 4 kinds of constraints are added to the model in order to make it operational for all the branch nodes throughout the branch-and-bound procedure. We also propose a new branching strategy for the ALP, which makes the branch-and-bound more efficient.

1.3 Outline of the thesis

The remainder of the thesis is structured as follows: In Chapter 2 more details about the ALP are introduced and a mathematical model for the ALP is presented. Furthermore a review of relevant literature is given. In Chapter 3, we reformulate the ALP as a set partitioning model and propose a branch-and-price method to solve the problem, in which the linear relaxation of the integer problem is solved by column generation for determining the lower bound, and the branch-and-bound method is

used to guarrantee the optimal integer solution for the ALP. A mathematical fomulation for the subproblem in the column generation is also proposed. In Chapter 4, an algorithm based on the branch-and-price method is developed. It is implemented and tested on the public data from OR-Library involving up to 50 aircraft. Computational results and data analysis are also presented. In Chapter 5, we summarize the achievements in this thesis and point out the future research on this work.

Chapter 2

The Aircraft Landing Problem

In this chapter, the description of the ALP is introduced. A mixed integer formulation for the ALP is presented. Finally, we review some recent approaches for the ALP in the literature, including genetic algorithm, population algorithm and so on.

2.1 Problem Description

Given a set of planes with target landing times and time windows for landings and runways, the objective of the ALP is to minimize the total (weighted) deviation from the target landing time for each plane. There are costs associated with landing either earlier or later than a target landing time for each plane. Each plane has to land on one of the runways within its predetermined time windows such that separation criteria between all pairs of plane are satisfied. This type of problem is a large-scale optimization problem, which occurs at busy airports where making optimal use of the bottleneck resource (the runways) is crucial to keep the airport operating smoothly.

Upon entering within the radar range (or horizon) of an air traffic

control (ATC) at an airport, a plane requires ATC to assign a landing time and also a runway if more than one runways are in use. The landing time must lie within a predetermined time window, bounded by an earliest landing time and a latest landing time. The time windows are different for different planes. The earliest time represents the time required if a plane flies at its maximum airspeed. The latest time corresponds to the landing time of a plane flying at its most fuel efficient airspeed while holding (circling) for the maximum allowable time (see Abela *et al.* (1993)).

Each plane also has a most economical, preferred speed, referred to as the cruise speed. The preferred or target time of a plane is the time it would land if it is required to fly at cruise speed. If ATC requires the plane to either slow down, hold or speed up, a cost will be incurred. Fig.2.1. depicts this variation in cost within the time windows of a plane.

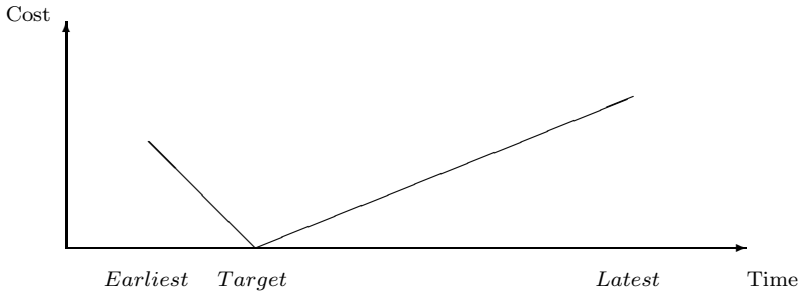

Figure 2.1: Variation in cost for a plane within its time window

Furthermore, the flow of incoming aircraft is not homogenous, it contains different aircraft types. All aircraft in flight create wake vortices at the rear of the craft. These vortices have a chaotic evolution and can cause serious turbulence to a closely following aircraft, even to the extent of a crash. In order to maintain an aircraft's aerodynamic stability, a separation distance based on the preceding aircraft types must be respected during landing. The separation time between two aircraft depends on the type of the aircraft, e.g. a Boeing 747 can

handle (and generates) more turbulence than a Hawker 700.

During peak hours, ATC must handle safely and effectively landings of a continuous flow of aircraft entering the radar range to the assigned runway(s). The capacity of the runways is highly constrained and this makes the scheduling of landings a difficult task to perform effectively. We might expect that the practical problem of scheduling aircraft landings within an ATC environment is more complex than the basic problem described above. We do not have perfect (or exact) information about all planes that are going to land. In practice the operational environment changes as time passes. New information becomes available making it necessary to revise the previous decision based on available information.

In the following, we consider the static (or off-line) version of the problem, where the set of aircraft waiting to land is known. This is in particular useful to investigate airport runway capacity in the planning stage. As customary in the big airports in practical case, there are usually more than one runways (e.g. the Copenhagen airport has 3 runways), therefore, we mainly discuss the multiple runway ALP in this work. Note that whilst throughout this thesis we shall only refer to planes landing, the formulation and algorithm presented here is applicable without change to problems also involving takeoffs.

2.2 A mathematical model of the ALP

This section presents a mixed integer formulation of the static multiple runway aircraft landing problem based on the formulaiton presented in Beasley *et al.* (2000).

Given a set of planes P, each plane i has a predetermined landing time windows $[E_i, L_i]$, and also, a target time T_i ($E_i \leq T_i \leq L_i$) at which time the plane is landed with cost 0. S_{ij} is the required separation time between plane i and j (where i lands before j) for landing these on the

same runway. As customary in the multiple runway case, we assume that the separation time between two planes on different runways is 0. g_i and h_i denote the unit costs for plane i landing earlier and later than the target time respectively.

We use the decision variables:

$$
\begin{aligned}
x_i &= \text{the landing time for plane } i \ (i \in P); \\
\alpha_i &= \text{how soon plane } i \text{ lands before } T_i \ (i \in P) \\
\beta_i &= \text{how late plane } i \text{ lands after } T_i \ (i \in P) \\
\delta_{ij} &= \begin{cases} 1 & \text{if plane } i \text{ lands before } j \ (i, j \in P; i \neq j); \\ 0 & \text{otherwise} \end{cases} \\
z_{ij} &= \begin{cases} 1 & \text{if } i \text{ and } j \text{ land on the same runway } \ (i, j \in P; i \neq j); \\ 0 & \text{otherwise} \end{cases} \\
y_{jr} &= \begin{cases} 1 & \text{if plane } j \text{ lands on runway } r \ (j \in P; r \in R); \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

The problem is to determine the landing time x and the allocation variable y for each plane which gives the minimum cost while satisfying the following:

- each plane lands at some time within the corresponding time window

$$x_i \in [E_i, L_i] \quad \forall i \in P; \tag{2.2.1}$$

- separation criteria between the landing of a plane, and the landing of all successive planes on the same runway are respected. That is, if $\delta_{ij} = 1$ and $z_{ij} = 1$, then the following holds,

$$x_j \geq x_i + S_{ij} \quad \forall i, j \in P; i \neq j \tag{2.2.2}$$

The mathematical formulation can now be stated as:

MIP:

$$\min \quad \sum_{i=1}^{P}(g_i\alpha_i + h_i\beta_i) \tag{2.2.3}$$

$$\text{s.t.} \quad E_i \leq x_i \leq L_i \qquad \forall i \in P; \tag{2.2.4}$$

$$\delta_{ij} + \delta_{ji} = 1 \qquad \forall i, j \in P; i \neq j \tag{2.2.5}$$

$$\sum_{r=1}^{R} y_{ir} = 1 \qquad \forall i \in P; r \in R \tag{2.2.6}$$

$$z_{ij} = z_{ji} \qquad \forall i, j \in P; i \neq j \tag{2.2.7}$$

$$z_{ij} \geq y_{ir} + y_{jr} - 1 \qquad \forall i, j \in P; i \neq j \tag{2.2.8}$$

$$x_j \geq x_i + S_{ij}z_{ij} - (L_i + S_{ij} - E_j)\delta_{ji} \forall i, j \in P; i \neq j \tag{2.2.9}$$

$$\alpha_i \geq T_i - x_i \qquad \forall i \in P \tag{2.2.10}$$

$$0 \leq \alpha_i \leq T_i - E_i \qquad \forall i \in P \tag{2.2.11}$$

$$\beta_i \geq x_i - T_i \qquad \forall i \in P \tag{2.2.12}$$

$$0 \leq \beta_i \leq L_i - T_i \qquad \forall i \in P \tag{2.2.13}$$

$$x_i = T_i - \alpha_i + \beta_i \qquad \forall i \in P \tag{2.2.14}$$

$$x_i, \alpha_i, \beta_i \geq 0 \qquad \forall i \in P \tag{2.2.15}$$

$$\delta_{ij}, y_{ij}, z_{ij} \text{ binary} \qquad \forall i, j \in P; i \neq j \tag{2.2.16}$$

The objective function (Eq.2.2.3) minimize the sum of the costs of deviation from the target times (T_i). Constraints (Eq.2.2.4) ensure that each plane lands within its time windows. Constraints (Eq.2.2.5) indicate that either plane i must land before plane j ($\delta_{ij} = 1$) or plane j must land before plane i ($\delta_{ji} = 1$). Constraints (Eq.2.2.6) ensure that each plan lands on exactly one runway whereas constraints (Eq.2.2.7) are symmetry constraints (if i and j land on the same runway so do j and i). Constraints (Eq.2.2.8) ensure that, if there is any runway r on which plane i and j are both landed (i.e. $y_{ir} = y_{jr} = 1$), then we force z_{ij} to be 1 (i and j land on the same runway). If $z_{ij} = 0$, then

(Eq.2.2.8) becomes $0 \geq y_{ir} + y_{jr} - 1$, ensuring that planes i and j can not land on the same runway. Constraints (Eq.2.2.9) are the separation constraints for plane i and j. There are four cases to consider here:

a. If $z_{ij} = 0$ and $\delta_{ji} = 1$, which means i and j land on different runway and j lands before i,it becomes

$$x_j \geq x_i - (L_i + S_{ij} - E_j)$$

the same as

$$x_j - E_j \geq x_i - L_i - S_{ij}$$

which always holds since $x_j - E_j \geq 0$ and $x_i - L_i - S_{ij} \leq 0$.

b. If $z_{ij} = 0$ and $\delta_{ji} = 0$ (indicating j lands after i on a different runway), (Eq.2.2.9) becomes

$$x_j \geq x_i + 0 - 0$$

which is in accordance with $\delta_{ji} = 0$.

c. If $z_{ij} = 1$ and $\delta_{ji} = 1$ (showing that j lands before i on the same runway), (Eq.2.2.9) becomes

$$x_j \geq x_i + S_{ij} - (L_i + S_{ij} - E_j)$$

the same as

$$x_j - E_j \geq x_i - L_i$$

which always holds since the left handside is always larger than or equal to zero while the right handsize is non-positive.

d. If $z_{ij} = 1$ and $\delta_{ji} = 0$ (showing that j lands after i on the same runway), (Eq.2.2.9) becomes

$$x_j \geq x_i + S_{ij}$$

ensuring the separation time between i and j must be fulfilled.

Constraints (Eq.2.2.10) and (Eq.2.2.11) ensure that α_i is at least as big as zero and the time difference between T_i and x_i, and at most the time difference between T_i and E_i. Constraints (Eq.2.2.12) and (Eq.2.2.13) are similar equations for β_i. Constraits (Eq.2.2.14) relate the landing time (x_i) to the time plane i lands before (α_i), or after (β_i), target (T_i).

This formulation is a mixed integer program involving $3P$ continous variables, $o(P^2)$ binary vaiables. The linear programming relaxation (denoted as **LMIP**) is obtained by relaxing the integer constraints (Eq.2.2.16) to

$$\delta_{ij}, z_{ij}, y_{ir} \in [0,1] \qquad \forall i,j \in P; i \neq j \tag{2.2.17}$$

There are a number of extensions to the above formulation of the ALP worth mentioning. Note first here that the objective is to minimize the total weighted deviation of the landing time from the target time. However, different objective functions can be adopted relating to practical considerations. For example, if we were using the model strategically to obtain some indication of the runway capacity, then we might use the objective function

$$min \quad max[x_i | i = 1, ..., P] \tag{2.2.18}$$

to land all the P planes as soon as possible. In practice, it could also happen that certain aircraft can not land on certain runways because, for example, a particular runway is under maintenance or it is too short for landing the aircraft. This can be easily dealt with by forcing $z_{ir} = 0$ if plane i can not use runway r.

2.3 Review of previous literature

With the problem of increasing congestion at airports, the efficient and effective scheduling of plane takeoffs and landings becomes very important. More and more work has been done recently in investigating the

ALP. Both heuristic methods and optimal methods have been developed to solve the ALP including simple heuristic, population heuristic, genetic algorithm etc.

Beasley *et al.* (2000) present a mixed integer formulation of the ALP and a detailed review of published work addressing the ALP. They propose 6 kinds of additional constraints in order to reduce the zero-one space of the mixed integer formulation. The problem is then solved optimally by using linear programming-based tree search for the public data from OR-Library involving up to 50 aircraft. An effective heuristic algorithm is also presented.

Ernst *et al.* (1999) present a specialized simplex algorithm which evaluates the landing time very rapidly, based on some partial order information. This method is then used in a problem space search heuristic as well as a branch-and-bound method for both single and multiple runway ALP. Preprocessing steps involving time window tightening and partial ordering based on problem data or an upper bound are used. The algorithm is tested by the instances from OR-Library involving up to 50 aircraft and 4 runways.

Jung *et al.* (2003) propose a heuristic algorithm based on the segmentation of time. The time horizon is divided into time segments that determine subproblems of ALP. Each subproblem is formulated as a mixed integer zero-one linear problem as in Beasley *et al.* (2000) and solved optimally in turn. Computational results are presented for instances from OR-Library and for randomly generated instances involving up to 75 aircraft and 4 runways.

Cheng *et al.* (1999) develop four different genetic-search formulations for the multiple runway ALP. Three of these schemes use a genetic algorithm approach while the last scheme uses a genetic programming approach. Computational results are presented involving 12 aircraft and 3 runways.

Pinol *et al.* (2005) first apply the scatter search and bionomic algorithm for the multiple runway ALP. The initial population consists of

three heuristic individuals based on the order of non decreasing earliest, target and latest time. The fitness value is defined as the objective function value while the unfitness value is measured by the violation of the separation time constraints. In the scatter search, binary tournament selection scheme based on individual fitnesses is used for parents selection. For each aircraft, the new proportion value is computed as a weighted linear combination of the corresponding parent propotion values. Random weights are used here in order to introduce diversity to the new individual. Furthermore, a duplication test is used to maintain a good level of diversity in the population. Then a simple linear program on the landing sequence with the order fixed is applied to improve the new individual. Afterwards, the new child is inserted into the population, and the current worst individual is removed in order to keep the population size constant. Both the linear and non-linear objective function is considered in their paper. Computational results are presented involving up to 500 aircraft and 5 runways.

Psaraftis *et al.* (1978, 1980) and Storer *et al.* (1992) adopt a job-shop scheduling approach for solving a version of ALP. The runways represent identical machines and the planes represent jobs. The earlist time associated with each plane is the ready time (or release time) of the job. Typically such papers assume the latest time to be sufficiently large to be of no consequence. The separation time in the ALP is considered as the processing time in job-shop scheduling. The processing time of a particular job (plane) on a particular machine (runway) is then dependent upon just the job following it on the same machine (successive separation), or all the other jobs that will follow it on the same machine (complete separation).

Bianco *et al.* (1993) also adopt a job-shop sheduling view of the ALP. They solve the single-runway problem using a mixed integer linear program and provide a tree-search procedure with embedded Lagrangean lower bounds. They also develop a heuristic approach for the problem.

The ALP may also be viewed as an open traveling salesman problem (TSP) with time windows when single runway and successive separation is considered. The difficulty with this approach lies in representing the objective function. Bianco *et al.* (1993) apply this method and develop dynamic programming algorithm for the TSP with cumulative costs. Bianco *et al.* (1999) also adopt this approach and present a dynamic programming formulation, lower bounds, and two heuristic algorithms. Computational results are presented for a number of randomly generated problems and as well as two problems from the OR_Library.

Among the heuristic algorithms, the simple heuristic (Beasley *et al.* (2000)) provides the fastest solutions. However, the solution quality is not stable. For the tested instances, the worst solution is around 77% away from the optimum. The time segment heuristic provides much better solutions. For the same instances, the optimality gap is less than 6.5%. Another good heuristic method is the population heuristic method. Moreover, it is very efficient and has been used to solve the problem involving up to 500 aircraft that are much larger than those in most of the papers.

All the exact algorithms (e.g. Beasley *et al.* (2000) and Ernst *et al.* (1999)) use the branch-and-bound method to search for optimal integer solution for the ALP. However, by using this method, the running time grows exponentially in the problem size. Hence, it is not able to solve the very large scale problems optimally within in acceptable time. In the literature, the ALP are solved to optimality up to 50 aircraft.

Chapter 3

The Branch-and-Price Method for ALP

In this chapter, we reformulate the ALP as a set partitioning model, of which the linear relaxation provides a good bound of the optimal solution. The preprocessing is introduced in order to reduce the solution space. A heuristic method for determining an upper bound, which is used to tighten the time windows, is presented. Finally, a branch-and-price method is proposed for solving the ALP.

3.1 A set partitioning model of the ALP

The experimental results presented in Beasley *et al.* (2000) show that the linear relaxation **LMIP** provides a poor bound on the optimal value of the mixed integer model **MIP** presented. In order to get a better formulation, we reformulate it as a set partitioning formulation. Planes covered by the same column are to be landed on the same runway. Additionally, side constraints on the number of available runways need to be added. Let S denote the set of all feasible landing sequences. Let a_i^s be 1 if plane i appears in the landing sequence s ($s \in S$) and 0 otherwise. Let c_s be the cost of the landing sequence s, which is given

by,

$$c_s = \sum_{i=1}^{P} (g_i \alpha_i a_i^s + h_i \beta_i a_i^s)$$

The resulting model becomes:

SP:

$$\min \quad \sum_{s \in S} c_s z_s \tag{3.1.1}$$

$$\text{s.t.} \quad \sum_{s \in S} a_i^s z_s = 1 \quad \forall i \in P \tag{3.1.2}$$

$$\sum_{s \in S} z_s = R \tag{3.1.3}$$

$$z_s \text{ binary} \tag{3.1.4}$$

where the z_s is the binary variable which is 1 if the landing sequence s is selected and 0 otherwise. (Eq.3.1.1) is the objective function minimizing the accumulated costs of all the planes. Constraints (Eq.3.1.2) ensure that each plane lands on exactly one runway, while the constraint (Eq.3.1.3) shows the limit on the number of the runways. Constraints (Eq.3.1.4) are the integrality constraints on the decision variable z_s. This is an integer program denoted as **SP**. Fig.3.1 shows an small instance of landing 3 planes on 2 runways. The time windows and the unit costs for the landings are given. The separation time between any of two planes is set to be 10.

Fig.3.2 shows the feasible landing sequences, corresponding landing time and the corresponding costs. Note here that the feasible landing sequences must fulfill the time windows, separation constraints and so on. Consider the landing sequence $\{2 \rightarrow 1\}$. Regarding the separation constraints between plane 2 and plane 1 (i.e. $S_{21} = 10$), the earliest time of landing plane 1 after the landing of plane 2 is 98 ($= E_2 + S_{21}$), which exceeds its time window ($[50, 95]$). Hence, $\{2 \rightarrow 1\}$ is not included in the solution space S of the set partitioning model. Furthermore, the *'total cost'* is the minimum cost for each landing sequence and

plane	E_i	T_i	L_i	g_i	h_i
1:	50	88	95	3	1
2:	88	95	105	3	1
3:	75	100	120	3	1

Figure 3.1: A small example of landing problem.

the $'landing\ time'$ is the optimal landing time corresponding to the minimum cost. This is because the objective of the ALP is to minimize the total cost. If a feasible sequence s is selected in the optimal solution, the planes involved in this sequence will land on the optimal landing time, and thus the total cost of the landings is minimized.

Consider the problem of finding the minimum cost and the optimal landing time for a feasible sequence s. This can be viewed as a single landing problem given a set of planes and also the order of the planes. Let P_s denote the set of planes appearing in sequence s and U_s denote the set of ordered pairs. For instance, for the feasible sequence 10 ($\{1 \rightarrow 3 \rightarrow 2\}$) in Fig.3.2, the corresponding P_{10} is $\{1, 2, 3\}$ and U_{10} is $\{(1, 3), (1, 2), (3, 2)\}$.

The objective is to find the landing time x_i ($\forall i \in P_s$) such that

1. $x_i \in [E_i, L_i]$
2. $x_j \geq x_i + S_{ij} \qquad \forall (i, j) \in U_s$
3. the sum of the weighted deviation from the target time is minimized

The notations α_i and β_i are used to denote the earliness and tardiness. The mathematical model can be formulated as

	landing sequence	*landing time*	*total cost*
sequence 1:	{1}	{88}	0
sequence 2:	{2}	{95}	0
sequence 3:	{3}	{100}	0
sequence 4:	{1→ 2}	{88 98}	3
sequence 5:	{1→ 3}	{88 100}	0
sequence 6:	{3→ 1}	{85 95}	52
sequence 7:	{2→ 3}	{95 105}	5
sequence 8:	{3→ 2}	{95 105}	25
sequence 9:	{1→ 2 → 3}	{88 98 108}	11
sequence 10:	{1→ 3 → 2}	{85 95 105}	34

Figure 3.2: The feasible sequences of Fig.3.1

SEQ:

$$\min \quad \sum_{i \in P_s} (g_i \alpha_i + h_i \beta_i) \tag{3.1.5}$$

$$\text{s.t.} \quad E_i \le x_i \le L_i \qquad \forall i \in P_s \tag{3.1.6}$$

$$x_j \ge x_i + S_{ij} \qquad \forall (i,j) \in U_s \tag{3.1.7}$$

$$\alpha_i \ge T_i - x_i \qquad \forall i \in P_s \tag{3.1.8}$$

$$0 \le \alpha_i \le T_i - E_i \qquad \forall i \in P_s \tag{3.1.9}$$

$$\beta_i \ge x_i - T_i \qquad \forall i \in P_s \tag{3.1.10}$$

$$0 \le \beta_i \le L_i - T_i \qquad \forall i \in P_s \tag{3.1.11}$$

$$x_i = T_i - \alpha_i + \beta_i \qquad \forall i \in P_s \tag{3.1.12}$$

$$x_i, \alpha_i, \beta_i \ge 0 \qquad \forall i \in P_s \tag{3.1.13}$$

The constraints (Eq.3.1.6) ensure that each plane appearing in P_s lands within its time window. Constraints (Eq.3.1.7) ensure that the separation time between i and j is larger than or equal to S_{ij} ($\forall(i,j) \in U_s$), since i lands before j in the landing sequence. Constraints (Eq.3.1.8 – Eq.3.1.12) are similar to the constraints (Eq.2.2.10 – Eq.2.2.15) in **MIP** as shown in chapter 2, related to the α_i, β_i and x_i. This is a linear program (denoted as **SEQ**) that can be used to find the optimal solution for any given landing sequences.

With the information of feasible landing sequences and the corresponding cost, the ALP problem can be formulated as a set partitioning problem. The corresponding set partitioning model of this small instance is shown in Fig.3.3.

$$
\begin{array}{lccccccccc}
\min & 0z_1 & +0z_2 & +0z_3 & +3z_4 & +0z_5 & +5z_6 & +11z_7 & \\
 & z_1 & & & + z_4 & + z_5 & & + z_7 & =1 \\
 & & z_2 & & + z_4 & & + z_6 & + z_7 & =1 \\
 & & & z_3 & & + z_5 & + z_6 & + z_7 & =1 \\
 & z_1 & + z_2 & + z_3 & + z_4 & + z_5 & + z_6 & + z_7 & =2
\end{array}
$$

Figure 3.3: The set partition formulation of Fig.3.1

Note here that the column in the formulation does not state the information of the order of the landings. For instance, for the last column $[1\ 1\ 1]'$ in the set partitioning model, there exist two feasible landing sequences including sequence 9 ($\{1 \rightarrow 2 \rightarrow 3\}$) with cost 11 and sequence 10 ($\{1 \rightarrow 3 \rightarrow 2\}$) with cost 34, as shown in Fig.3.2. However, because that the objective of the ALP is to minimize the total cost, if the last column is selected, it is obvious that the planes

will land in the order of sequence 10 which has the minimum cost among the costs of the two feasible sequences corresponding to this column. Therefore, the coefficient in the objective function should be the minimum cost of landing the planes appearing in the corresponding column. For the small example, we enumerate all the feasible sequences corresponding to the column, calculate the costs for these sequences and choose the minimum cost to be the coefficient (called enumeration method). However, for large-scale problems, it is time consuming to use the enumeration method, since there exist too many feasible sequences for the columns. Instead, a mathematical model can be formed to determine the minimum cost for a column. This optimization problem is slightly different formulated than determining the minimum cost for a given landing sequence, since the order of the planes is unknown here. Based on the **SEQ**, this model can be obtained by adding an additional variable δ_{ij} which is 1 if plane i lands before plane j and 0 otherwise, the constraints (Eq.3.1.14) to ensure that plane i lands either before or after plane j, and the separation time constraints (Eq.3.1.15).

$$\delta_{ij} + \delta_{ji} = 1 \qquad \forall i, j \in P_a; i \neq j \tag{3.1.14}$$

$$x_j \geq x_i + S_{ij}\delta_{ij} - (L_i - E_j)\delta_{ji} \qquad \forall i, j \in P_a; i \neq j \tag{3.1.15}$$

Let P_a denote the set of planes appearing in the column a. The complete model can be obtained as follows:

COL:

$$\min \quad \sum_{i \in P_a} (g_i \alpha_i + h_i \beta_i) \tag{3.1.16}$$

$$\text{s.t.} \quad E_i \leq x_i \leq L_i \qquad \forall i \in P_a; \tag{3.1.17}$$

$$\delta_{ij} + \delta_{ji} = 1 \qquad \forall i, j \in P_a; i \neq j \tag{3.1.18}$$

$$x_j \geq x_i + S_{ij}\delta_{ij} - (L_i - E_j)\delta_{ji} \qquad \forall i, j \in P_a; i \neq j \tag{3.1.19}$$

$$\alpha_i \geq T_i - x_i \qquad \forall i \in P_a \tag{3.1.20}$$

$$0 \leq \alpha_i \leq T_i - E_i \qquad \forall i \in P_a \tag{3.1.21}$$

$$\beta_i \geq x_i - T_i \qquad \forall i \in P_a \tag{3.1.22}$$

$$0 \leq \beta_i \leq L_i - T_i \qquad \forall i \in P_a \tag{3.1.23}$$

$$x_i = T_i - \alpha_i + \beta_i \qquad \forall i \in P_a \tag{3.1.24}$$

$$x_i, \alpha_i, \beta_i \geq 0 \qquad \forall i \in P_a \tag{3.1.25}$$

$$\delta_{ij} \text{ binary} \qquad \forall i, j \in P_a; i \neq j \tag{3.1.26}$$

This is a mixed integer formulation (denoted as **COL**) used to determine the minimum cost, the landing sequence and the landing time for the columns in the set partitioning model. Fig. 3.4 shows the complete information of the set partitioning model for this small instance.

The ALP can be solved by solving the set partitioning model. For the small example, as shown in Fig.3.3, the optimal solution is $Z^* = [0, 1, 0, 0, 1, 0, 0]$. According to the information shown in Fig.3.4, the corresponding optimal landings for the ALP can be obtained as shown in table 3.1.

variable	c_s	a'_s	*landing sequence*	*landing time*
z_1	0	[1 0 0]	{1}	{88}
z_2	0	[0 1 0]	{2}	{95}
z_3	0	[0 0 1]	{3}	{100}
z_4	3	[1 1 0]	{1→ 2}	{88 98}
z_5	0	[1 0 1]	{1→ 3}	{88 100}
z_6	5	[0 1 1]	{2→ 3}	{95 105}
z_7	11	[1 1 1]	{1→ 2 → 3}	{88 98 108}

Figure 3.4: The complete information for the set partitioning model.

3.2 Preprocessing

In order to make the calculation more efficient, the formulation is tightened before we start to solve the optimization problem. This can be done by fixing some variables, reducing the feasible interval of some variables etc. Thereby, the solution space is narrowed.

In the ALP, the predetermined landing time window for each plane depends on the airspeed at which the plane flies. As mentioned above,

runway	landing column	landing sequence	cost
1	[0 1 0]	{2}	0
2	[1 0 1]	{1 → 3}	0
Total Landing Cost			0

Table 3.1: Final results for the small example in Fig.3.1

the earliest time is the landing time of the plane flying at its maximum airspeed while the latest time is at its most fuel-effiicient airspeed. Due to the difference of the airspeed, a plane is allowed to land in a very wide time window without considering the total cost. However, in practical, the cost has to be considered. For example, in practical airline operations, a plane would never land at its latest landing time since this costs too much although it is feasible. This makes the predetermined latest time to be of no consequence. Moreover, the overlapes of the allowable landing time of two different planes will also be large intervals with the $'$loose$'$ time windows. This will result in a huge number of feasible solutions. For example, we have two planes 1 and 2 with time windows of $[50, 571]$ and $[200, 760]$, respectively, and we assume the separation time is $S_{12} = 20$ and $S_{21} = 30$. We can see that plane 1 can be landed either before or after plane 2 (i.e $\delta_{12} = 1$ or $\delta_{21} = 1$). In other words, landing sequences $\{1 \rightarrow 2\}$ and $\{2 \rightarrow 1\}$ are both feasible here, although $\{2 \rightarrow 1\}$ may incur a very large cost.

Suppose we already know the upper bound of the cost (denoted as Z_{UB}) which will definitely be large or equal to the optimal solution. Then it is possible to limit the deviation from target time for each plane. For plane i, if we assume that all other planes make a zero contribution to the objective function value, we can update E_i using

$$E_i = max[E_i, T_i - Z_{UB}/g_i] \qquad \forall i \in P \qquad (3.2.1)$$

because the cost would exceed the Z_{UB} if we land i more than Z_{UB}/g_i time units before its target time. Similarly, for the latest time, we have that

$$L_i = min[L_i, T_i + Z_{UB}/h_i] \qquad \forall i \in P \qquad (3.2.2)$$

By using equations (Eq.3.2.1) and (Eq.3.2.2), the time windows $[E_i, L_i]$ for each plane i ($\forall i \in P$) can be tightened. Assume that, for the above two planes, the time windows becomes $[57, 80]$ and $[200, 230]$ respectively. Note here that sequence $\{2 \rightarrow 1\}$ is not feasible any

more. Therefore, the value of δ_{12} is fixed to be 0 and δ_{21} to be 1. The solution space is hence reduced. The better the upper bound is, the more significant the solution space is reduced, especially for the large-scale problems.

As we known, for a minimize problem, any feasible solution can form an upper bound of the optimal solution. Hence, a general way to provide an upper bound is searching for a feasible solution which can be done through an approximate algorithm or heuristic method. In this thesis, a simple heuristic method based on Beasley *et al.* (2000) is used to determining the upper bound for the total cost of the given landing problem.

Specifically, the planes are ordered in nondecreasing target time, then assigned one by one to the runway with the least cost. The landing cost on a runway is calculated according to the previous landings and the corresponding separation time. Let B_{ir} be the cost of airplane i landing on runway r, then

$$B_{ir} = max_{k \in A_r}\{T_i, x_k + S_{ki}\}$$

where A_r is the previous landings on runway r and plane i is the one being considered for a runway assignment. The current plane is assigned to the runway r^* with the minimum B_{ir}. Hence, after this step, the temporary assignment of landing times are always on or after the target times. To improve the solution obtained, the LP problem with the fixed order of landing on the runway is solved.

3.3 Branch-and-Price

Branch-and-price is known to be an efficient method for solving large-scale scheduling problems such as the vehicle routing problem, the crew scheduling problem etc. However, it has not been applied for the ALP in the literature. In this section, we propose a branch-and-price method for the ALP.

3.3.1 Column Generation

For the small example involving 3 planes shown in Fig.3.1 involving 3 planes, it is possible to enumerate all possibilities of the landing sequences as shown in Fig.3.1. However, for a problem with 50 planes, there exsit $\sum_{k=1}^{50} \binom{50}{k} \approx 1.1259 \times 10^{15}$ feasible columns. It is computational inefficient to enumerate all these columns. We can start by solving a restriced problem which is an LP relaxation with only a small subset of the the variables (called *master problem*). In other words, the value of the rest variables are fixed to be 0. Then, we check if the addition of one or more variables, currently not in the restricted linear program, might improve the LP-solution. According to the simplex algorithm, this can be achieved by adding the variable with negtive reduced cost. The problem of finding a variable with negative reduced cost is called the *subproblem*. The reduced cost is defined as:

$$\widetilde{c}_j = c_j - \pi a \qquad \forall j \in S$$

where c_j is the cost coefficient for column a, π is the dual values corresponding to each constraints of the linear system provided by the optimal basic solution of the master problem.

This method of solving linear program is called column generation. It has been demonstrated to be a successful method for solving the linear program with huge number of variables.

In our case, the linear relaxation program (denoted as **LSP**) can be obtained from the integer program (**SP**) without the integrality constraints (Eq.3.1.4). The master problem is initiated with a set of columns that is generated during the initilizaiton. The subproblem is to find the column with the minimum reduced cost. A mathematial formulation for the subproblem is proposed and shown as following:

SUB:

$$\min \quad \sum_{i=1}^{P}(g_i\alpha_i + h_i\beta_i) - \sum_{i=1}^{P}\pi_i a_i - \lambda \tag{3.3.1}$$

$$\text{s.t.} \quad a_i E_i \leq x_i \leq a_i L_i \qquad \forall i \in P \tag{3.3.2}$$

$$\delta_{ij} \leq a_i \qquad \forall i, j \in P; i \neq j \tag{3.3.3}$$

$$\delta_{ij} \leq a_j \qquad \forall i, j \in P; i \neq j \tag{3.3.4}$$

$$1 \geq \delta_{ij} + \delta_{ji} \geq a_i + a_j - 1 \qquad \forall i, j \in P; i \neq j \tag{3.3.5}$$

$$x_j \geq x_i + S_{ij}\delta_{ij} - (L_i - E_j)\delta_{ji} - (a_i L_i - a_j E_j) + (L_i - E_j)(\delta_{ij} + \delta_{ji}) \qquad \forall i, j \in P; i \neq j \tag{3.3.6}$$

$$\alpha_i \geq a_i T_i - x_i \qquad \forall i \in P \tag{3.3.7}$$

$$0 \leq \alpha_i \leq T_i - E_i \qquad \forall i \in P \tag{3.3.8}$$

$$\beta_i \geq x_i - a_i T_i \qquad \forall i \in P \tag{3.3.9}$$

$$0 \leq \beta_i \leq L_i - T_i \qquad \forall i \in P \tag{3.3.10}$$

$$x_i = a_i T_i - \alpha_i + \beta_i \qquad \forall i \in P \tag{3.3.11}$$

$$\delta_{ij}, a_i \text{ binary} \qquad \forall i, j \in P \tag{3.3.12}$$

where π_i denotes the dual variable value corresponding to plane i, for each $i \in P$, in constraint (Eq.3.1.2), and λ denotes the dual variable value corresponding to the (Eq.3.1.3) in the master problem. a_i is the binary variable which is 1 if plane i appears in the landing sequence and 0 otherwise. This can be considered as a single runway landing problem for a subset of the planes.

The landing time (x_i), earliness (α_i) and tardiness (β_i) are active only if the plane i appears in the landing sequence (i.e. $a_i = 1$). This is ensured by constraints (Eq.3.3.2). Similarly, constraints (Eq.3.3.3) and (Eq.3.3.4) show that δ_{ij} is active if i and j both are in the landing sequence (i.e. $a_i = a_j = 1$). Constraints (Eq.3.3.5) show that either plane i must land before plane j ($\delta_{ij} = 1$) or plane j must land before

plane i ($\delta_{ji} = 1$) given both of them are active (i.e. $a_i = a_j = 1$). Constraints (Eq.3.3.6) enforce the separation time between two planes. There are 4 cases considered here:

a. If both of plane i and j are active (i.e. $a_i = a_j = 1$), then either i lands before j (i.e. $\delta_{ij} = 1$) or j lands before i (i.e. $\delta_{ij} = 0$). Therefore, (Eq.3.3.6) becomes either $x_j \geq x_i + S_{ij}$ ensuring that separation is enforced, or $x_j \geq x_i - (L_i - E_j)$ a constraint that is always true.

b. If $a_i = 1$ $a_j = 0$, from (Eq.3.3.3) and (Eq.3.3.4) we have that $\delta_{ij} = \delta_{ji} = 0$. From (Eq.3.3.2) we have $x_j = 0$ given $a_j = 0$. Therefore, (Eq.3.3.6) becomes $0 \geq x_i - L_i$, which is already covered by constraint (Eq.3.3.2).

c. If $a_i = 0$ $a_j = 1$, similar to case b, (Eq.3.3.6) becomes $x_j \geq E_j$.

d. If $a_i = 0$ $a_j = 0$, it becomes $0 \geq 0$.

The objective function (Eq.3.3.1) is the reduced cost of column a. The first term $\sum_{i=1}^{P}(g_i\alpha_i + h_i\beta_i)$ is the cost of the column.

Consider a plane that is not covered by the column (i.e. $a_i = 0$). Its landing time will simply be set to be 0 (i.e. $x_i = 0$) by (Eq.3.3.2). In this case, the constraint (Eq.3.3.7) becomes $\alpha_i \geq 0$ which is covered by the constraint (Eq.3.3.8). Thereby, we have $\alpha_i \in [0, T_i - E_i]$. Similaly, we have $\beta_i \in [0, L_i - T_i]$ by the constraints (Eq.3.3.9) and (Eq.3.3.10). Given $a_i = 0$ and $x_i = 0$, we can obtain $\alpha_i = \beta_i$ from constraint (Eq.3.3.11). Since the unit costs g_i and h_i in the objective function are positive, in the optimal solution, both α_i and β_i will be forced to be 0 by minimizing the objective function. In other words, the plane not appearing in the column will have no effect on the objective function coefficient (Eq.3.3.1) .

The subproblem is denoted as **SUB**. If the objective function value is non-negative, then the master problem **LSP** has been solved, otherwise the corresponding column a is added to the master problem and

the master problem is then solved again.

The general framework of column generation is presented in Fig.3.5.

---

generate an initial feasible solution (columns)

repeat

solve the master problem by linear programming

find out the columns with negative reduced cost

add the column(s) into the master problem

until termination, when there exists no column with negative reduced cost

---

Figure 3.5: The framework of column generation

3.3.2 Branch-and-Bound

In most cases, the solution of **LSP** is a fractional solution. In order to guarantee that we end up with an integer solution, the column generation method is combined with a branch-and-bound method (so called branch-and-price) in which the column generation provides the lower bound for each node throughout the exploration of branch-and-bound tree.

Branch-and-bound is a general search method. Suppose we wish to minimize a function $f(x)$, where x is restricted to some feasible region (defined, i.e. by explicit mathematical constraints). To apply branch and bound, one must have a means of computing a lower bound on the

optimization problem and a means of dividing the feasible region of the problem to create smaller subproblems. Moreover, there usually also have to be a way to compute an upper bound (e.g. a feasible solution).

The method starts by considering the original problem with the complete feasible region, which is called the root problem. The lower-bounding and upper-bounding procedures are applied to the root problem. If the bounds match, then an optimal solution has been found and the procedure terminates. Otherwise, the feasible region is divided into two or more regions, each is a strict subregion of the original, which together cover the whole feasible region. Ideally, these subproblems partition the feasible region. These subproblems become children node of the root search node. The algorithm is applied recursively to the subproblems, generating a tree of subproblems. The upper bound for the problem is updated if we find a feasible solution that is better than the current best solution, and it can be used to prune the rest of the tree: If the lower bound for a node exceeds the best known feasible solution, no globally optimal solution can exist in the subspace of the feasible region represented by the node. Therefore, the node can be removed from consideration. The search proceeds until all nodes have been solved or pruned.

In the following, we give some details of our branch-and-bound algorithm for solving the entire problem **SP**.

Bounding

In order to evaluate a given subspace, a bound value is computed. In our case, a lower bound is given by the linear relaxation problem **LSP** with some restriction on partial landing sequence sets S imposed by branching rules (described later). An upper bound (Z_{UB}), that is the minimum integer solution value abtained so far, is associated with the branch-and-bound tree. At each iteration, one branch-and-bound node is solved using the column generation approach described

previous. The restricted master problem is initialized using all the columns of its father node except the one that must be deleted based on branching rules. There are three possible cases for the LP solution to a branch-and-bound node.

Case 1: If the solution is integer, then we first prune this node from the branch-and-bound tree, since none of its offsprings will produce better integer solution. Then the solution value (Z_{SV}) is compared with the current upper bound (Z_{UB}) of the entire tree. If $Z_{SV} < Z_{UB}$, then this node becomes the best integer node and the upper bound of the problem is updated: $Z_{UB} := Z_{SV}$, and the lower bound of each active branch-and-bound tree node is compared with this new upper bound. And those nodes for which lower bound is larger than the Z_{UB} is not necessary to be considered any more and is thus pruned from the tree.

Case 2: If the solution is fractional and its solution value is greater than or equal to the upper bound Z_{UB}, then this node is pruned from the tree since the integer solutions of its offspings will be no better than the fractional solution.

Case 3: If the solution is fractional and its solution value is less than the Z_{UB}, then a branching decision is made to create two child nodes of this node based on the fractional solution.

Branching

According to the branch-and-bound method, if the solution of the current node satisfies Case 3, a branching decision has to be made to create two son nodes. A customary branching way is to branch on the binary decision variables in the model. In our formulation, the value of each variable z_s indicates the selection of the corresponding landing sequence s. For the ALP, $z_s = 0$ means that the partial landing sequence is excluded and hence no such sequence can be generated in subsequent subproblems. However, it is very hard to exclude a sequence when

solving a single runway problem.

In our algorithm, instead of branching on the z-variable, a new branching decision is constructed as follows: we branch on whether plane j is landed immediately after i or not. This is called Ryan-Foster branching (Ryan *et al.* (1981)) which has been used for solving crew scheduling and vehicle routing problems (Larsen (1999)). In our problem, let y_{ij} denote the new variable which is 1 if plane j is landed immediately after i and 0 otherwise. For any feasible solution of **LSP**, $(\bar{z}_s, s \in S)$, the corresponding $\bar{y}_{ij}$ is given by

$$\bar{y}_{ij} = \sum_{s \in S} w_{ij}^s \bar{z}_s \geq 0 \qquad \forall i, j \in P; i \neq j \tag{3.3.13}$$

where w_{ij}^s is 1 if $s \in S$ covers both plane i and plane j and plane i is immediately after plane j, and 0 otherwise.

Consider a branch-and-bound node and suppose its **LSP** solution (denoted as $\bar{z}_s, s \in S$), is fractional and is not pruned. Compute the corresponding $\bar{y}_{ij}$ value by (Eq.3.3.13). Then a branching decision can be made on this node. A pair (m, n) is selected such that $\bar{y}_{mn}$ is the fractional value closest to 1, i.e.$\bar{y}_{mn} = \max\{\ \bar{y}_{ij} | \bar{y}_{ij} \in (0, 1)\}$. Then two son nodes are created, one along the branch with $\bar{y}_{mn}$ fixed as 1 and the other as 0. Constraints enforcing this requirement need to be added to the problem.

1. If $\bar{y}_{mn}$ is fixed to 1, the initial restricted master problem of the corresponding child node consists of all the columns of its father node where plane n lands immediately after plane m or where none of these appear. For example, for an ALP involving 10 aircraft, the feasible columns for the branch node $\bar{y}_{23} = 1$ consist of the following two parts:

PartA. Neither of the two planes exist (i.e. $a_2 = a_3 = 0$). For example, planes land in the sequence $\{1 \rightarrow 5 \rightarrow 8 \rightarrow 10\}$.

PartB. Both of them are covered by the column (i.e. $a_2 = a_3 = 1$) and plane 3 lands immediately after plane 2. For example, planes land in the sequence $\{1 \rightarrow \mathbf{2} \rightarrow \mathbf{3} \rightarrow 6 \rightarrow 9 \rightarrow 10\}$

Therefore, the structure of the subproblem **SUB** must be updated. The following two constraints are imposed:

$$a_m = a_n \tag{3.3.14}$$

$$\sum_{k\in P} \delta_{km} = \sum_{k\in P} \delta_{kn} - a_m \tag{3.3.15}$$

Constraint (Eq.3.3.14) indicates that either both or none of them are covered by the landing sequence (i.e. either $a_m = a_n = 1$ or $a_m = a_n = 0$). In constraint (Eq.3.3.15), $\sum_{k\in P} \delta_{km}$ is the number of the planes that land before plane m, while $\sum_{k\in P} \delta_{kn}$ corresponds to plane n. There are two cases considered here

i. If $a_m = a_n = 0$, from (Eq.3.3.3) and (Eq.3.3.4) we have $\delta_{km} = \delta_{kn} = 0$ $(\forall k \in P)$. Therefore, (Eq.3.3.15) becomes $0 = 0 - 0$ which is always true.

ii. If $a_m = a_n = 1$, (Eq.3.3.15) becomes $\sum_{k\in P} \delta_{km} = \sum_{k\in P} \delta_{kn} - 1$ showing that plane n lands immediately after plane m, which is in accordance with $\bar{y}_{mn} = 1$

2. If $\bar{y}_{mn}$ is fixed to 0, the initial restricted master problem of the corresponding child node consists of all the columns of its father node except those where plane n is landed immediately after plane m. For the same example above, for the branch node with $\bar{y}_{mn} = 0$, the feasible columns consist of,

PartA. Neither of the two planes exist (i.e. $a_2 = a_3 = 0$). For example, planes land in the sequence $\{1 \to 5 \to 8 \to 10\}$.

PartB. Only one of them appears in the column (i.e. $a_2 = 1$ or $a_3 = 1$), e.g. $\{\mathbf{2} \to 9 \to 7 \to 10\}$ and $\{1 \to \mathbf{3} \to 9 \to 8 \to 10\}$.

PartC. Both of them are covered by the column (i.e. $a_2 = a_3 = 1$) but plane 3 does not land immediately after plane 2, e.g. $\{1 \to \mathbf{2} \to 9 \to \mathbf{3} \to 10\}$ and $\{7 \to \mathbf{3} \to \mathbf{2} \to 9 \to 4\}$.

Therefore, the following two constraints should be added to the subproblem:

$$a_m + a_n \leq 1 + M d_{mn} \tag{3.3.16}$$

$$2 - M\delta_{nm} \leq \sum_{k\in P} \delta_{kn} - \sum_{k\in P} \delta_{km} + M(1 - d_{mn}) \tag{3.3.17}$$

where d_{mn} is a binary variable. M is a large number. In this case, either constraint (Eq.3.3.16) or constraint (Eq.3.3.17) is active.

i. If $d_{mn} = 0$, (Eq.3.3.16) becomes $a_m + a_n \leq 1$ which satisfies PartA and PartB. From (Eq.3.3.3) and (Eq.3.3.4) in the **SUB**, we have $\delta_{nm} = 0$. Therefore, (Eq.3.3.17) becomes $2 - 0 \leq \sum_{k\in P} \delta_{kn} - \sum_{k\in P} \delta_{km} + M$ which always holds since M is sufficiently large.

ii. If $d_{mn} = 1$, (Eq.3.3.17) becomes $2 - M\delta_{nm} \leq \sum_{k\in P} \delta_{kn} - \sum_{k\in P} \delta_{km}$. If plane n lands before m (i.e. $\delta_{nm} = 1$), (Eq.3.3.17) becomes $2 - M\delta_{nm} \leq \sum_{k\in P} \delta_{kn} - \sum_{k\in P} \delta_{km}$ which always holds since M is sufficient large. If plane n lands after m (i.e. $\delta_{nm} = 0$), (Eq.3.3.17) becomes $2 \leq \sum_{k\in P} \delta_{kn} - \sum_{k\in P} \delta_{km}$ indicating that at least one plane lands after m and before n which is in accordance with the assumption.

With the 4 types of constraints (Eq.3.3.14–Eq.3.3.17) added into the **SUB**, the subproblem can be used to generate the feasible columns for each branch node throughout the branch-and-bound procedure.

Selection

As mentioned above, in each iteration, one node of the unexplored nodes is to be considered. Usually, the following three strategies are used to select the node: *Best-first Search* (BeFS) in which the node that has the lowest lower bound is selected; *Breath-first Search* (BFS) in which the branch-and-bound tree is explored level by level; *Depth-first Search* (DFS) in which the child node with the largest level is selected.

In our case, the node selection strategy used is the DFS. If the current branch-and-bound node is not pruned (i.e.the solution is fractional and is less than the upper bound Z_{UB} of the branch-and-bound tree), then the branching scheme is made on this node, and the child node with $\bar{y}_{mn} = 1$ is selected as the next node to be explored.

3.3.3 The overall method

To sum up, the framework of the entire branch-and-price method is depicted in Fig.3.6. The column generation is started with the generated columns in the initialization. More details about the initial columns will be discussed in section 3.4. When no more columns can be generated and the lower bound of the branch-and-bound node is determined, if the solution is integer, we compare it with the gloable upper bound Z_{UB}, then update the Z_{UB} and prune the current node. If it is fractional and lower than the gloable upper bound Z_{UB} then the node is pruned, otherwise it is divided into two child nodes. Then a new node is selected to be explored.

3.4 Implementation Details

3.4.1 Preprocessing

Tighten the time windows

As disscussed in section 3.2, a heuristic method is used to provide the upper bound of the problem. The main idea is to first generate a greedy solution in which the planes are sorted by the non-decreasing target time and are considered to be landed on the runway at its best time one by one. Then the solution is improved by solving the LP problem with the order of landing on the runway fixed. The implementation is

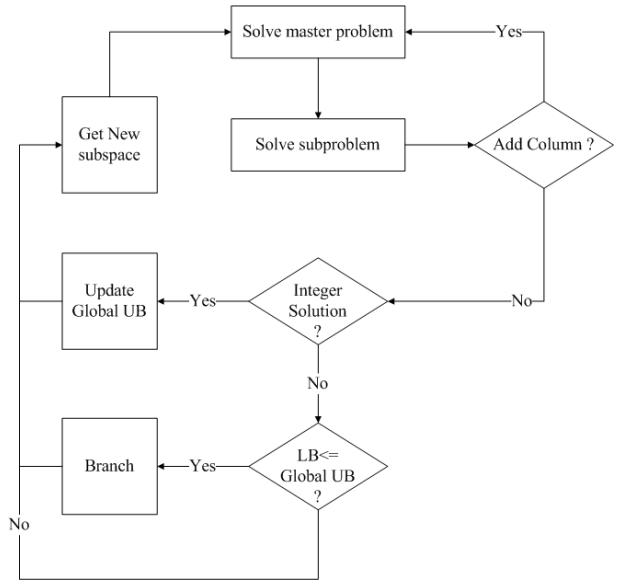

Figure 3.6: The overall structure of branch-and-price algorithm.

shown in Fig.3.7. Details see *airland_heuristic.m* in Appendix A.

Determine the cost coefficient for a column

As described in section 3.1, the coefficient in the objective function of the set partitioning model is the minimum cost of landing the planes appearing in the corresponding column. Two method for solving the problem are proposed: the enumaration method and the mathematical modeling method. The second one can be solved by GAMS. The enumeration method is described as following:

If there is only one plane covered in the column, it can be simply assigned to landing on its target time, and the cost will be 0.

If there are two planes i and j (assumed that $T_i < T_j$) appearing in the column, it is obvious that to land plane i before j is cheaper than the other way. Furthermore, if the separation time between i

initialization
sort the planes by the target time
for $j = 1$ to P
    for $r = 1$ to R
        B_{jr}:= the best time that j can be land on runway r
    end
    X_j:= min{ $B_{jr}|r = 1, ..., R$}
    $rway$:= the corresponding r to the min{ $B_{jr}|r = 1, ..., R$}
    A_{rway} := $[A_{rway}, j]$ (add the plane j on runway $rway$)
end
return X (the landing time for each plane)
if $X \neq T$
    X:= the solution of the LP problem with the fixed order
end
$Z_{UB} = \sum_{i=1}^{P} g_i \max(0, T_i - X_i) + h_i \max(0, X_i - T_i)$
return Z_{UB}

Figure 3.7: The heuristic method for Z_{UB}

and j is satisfied for landing both of them on their target time (i.e. $T_j - T_i > S_{ij}$), the column has the minimum cost that equals to 0, the corresponding landing sequence $\{i, j\}$ and landing time $\{T_i, T_j\}$. Otherwise, either plane i needs to speed up or plane j has to land after its target time, therefore, the minimun cost is $\min\{g_i, h_j\}(S_{ij} - T_j + T_i)$.

If there exist three or more planes in the column, then the problem becomes more complex. However, it is important to note that the cost of a complete sequence is always bigger than or equal to the cost of its partial sequence. It is therefore possible to quickly find a large number of sequences that have larger cost than the upper bound Z_{UB} by just checking the cost of their partial sequences. Based on this principle,

we start with landing only one plane. In each iteration, one new plane is inserted into each of the existing landing sequences, which contain the planes that have been considered in the previous iterations. The plane is allowed to be put into any position of the existing sequences if it is feasible (the time windows and separation time satisfied). We then calculate the costs for all feasible sequences and remove those with costs lager than the upper bound Z_{UB}. The rest of the sequences become the current sequences for the next iteration.

3.4.2 Columns Generation

Our implementation of generating the initial column is done by *ini_good_cols.m* in Matlab (see Appendix B). The linear programming master problem and subproblem is solved by the GAMS programs *air.gms* and *subproblem.gms* (refer to Appendix C).

Initial columns

Basically, the initial columns for our column generation method consist of two main parts. The first part is P dummy columns. Each column contains one plane without taking account of the runways, that is, the column contains a single 1 for the i'th row ($a_i^s = 1$). They are added to ensure a feasible LP upon branching and they are assigned a cost sufficiently high in order to force them out of the basis in the optimal solution. Fig.3.8 shows an example of the dummy column part in the master problem for a landing problem involving 4 aircraft and 2 runways. The second part is the columns corresponding to the heuristic solution obtained in the preprocessing. This is added in order to get a good starting objective value.

In order to improve the initial dual value of the master problem, we also do some experiments on adding some additional initial columns besides those initial columns mentioned above. The way we used to

$$
\begin{array}{lllllllllll}
\min & 0z_1 & + & 0z_2 & + & 0z_3 & + & 0z_4 & + & \cdots & \\
\\
\text{s.t.} & 1z_1 & + & 0z_2 & + & 0z_3 & + & 0z_4 & + & \cdots & =1 \\
& 0z_1 & + & 1z_2 & + & 0z_3 & + & 0z_4 & + & \cdots & =1 \\
& 0z_1 & + & 0z_2 & + & 1z_3 & + & 0z_4 & + & \cdots & =1 \\
& 0z_1 & + & 0z_2 & + & 0z_3 & + & 1z_4 & + & \cdots & =1 \\
& 0z_1 & + & 0z_2 & + & 0z_3 & + & 0z_4 & + & \cdots & =2
\end{array}
$$

Figure 3.8: The *dummy* part in the master problem for an small example

generate these columns is as follows: The planes are ordered in nondecreasing target time, then they are considered one by one whether to be landed or not. If the cost of landing the current plane is 0, in other words, the separation time between its target time and the previous landings is satisfied, we assign it to the runway, otherwise we discard it. Let k denote the number of non-existing planes. The next step is to generate k columns. In each column, we first fix one of the non-existing planes to be landed at its target time, then insert the rest P-1 planes as many as we can. By doing this, we ensure that our columns generated cover all the planes. The details of the algorithm for generating the columns are shown in Fig.3.9.

Master problem

As we mentioned in section 3.3.2, the column generation is applied for each of the branch-and-bound nodes. In branch-and-bound, the feasible

```
initialization
ordindT = sortindex(T);
cost0_col = zeros(num,P);
nonexist = [];
col0_col(1, ordindT(1)) = 1;
for j = 2 to P
    if {the separation time between T(ordindT(j))
       and the previous landings is satisfied}
         cost0_col(1, ordindT(j)) = 1;
    else
         nonexist = [nonexist, j];
    end
end
k=length(nonexist);
for i = 1 to k
    for j = 1 to P
         if {the separation time between T(ordindT(j))
            and the previous landings is satisfied}
              cost0_col(i+1,ordindT(j)) = 1;
         end
    end
end
return cost0_col[]
```

Figure 3.9: The generation of the initial columns.

region of each node is a subset of the original feasible region. Before we start the column generation for a node, we need to specify which columns of its father node are not feasible any more and remove them from the master problem. In our implementation, in order to avoid the recalculation of the same column, we do not remove them from our column set $Z[]$. We achieve this by doing the following: create a new variable fix corresponding to z and add a new constraint (Eq.3.4.1) in the master problem, initialize the variable fix to be [0,...,0] at the begining, set $fix(s)$ to be 1 if the column s is checked to be infeasible due to the branch-and-bound. With adding the constraits (Eq.3.4.1), it is easy to see that, the column s will never be selected in solving the current node, since the value of the variable z_s is forced to 0.

$$z_s = 0 \qquad \forall s \in S; \quad fix(s) = 1 \tag{3.4.1}$$

Consider the small example involving 3 planes and 2 runways as shown in section 3.1. Suppose the solution of the linear relaxation **LSP** is fractional and the child node with $y_{23} = 1$ is selected to be explored. From Fig.3.4, we can see the column 1, 6, and 7 are feasible for this node, while the column 2, 3, 4, and 5 are not feasible any more. By updating fix :=[0 1 1 1 1 0 0], the infeasible columns will never be selected in the solution of this node, since the values of the corresponding variables z_2, z_3, z_4, and z_5 are forced to be 0 by the constraints (Eq.3.4.1).

Subproblem

Similar to the master problem, the columns generated in the subproblem also need to fulfill the feasibility of the current branch node.Given the information of the branch node, the subproblem is only allowed to generate the columns lying in the feasible region of the current node. In our implementation, the column generating procedure in the subproblem also controlled by a matrix $node[]$ (P by P) which is initilized

to be 0 for all $node(i, j)$. Based on the $node[]$ of its father node, the corresponding $node(i, j)$ is set to be 1 for the branch of $y_{ij} = 1$ and $node(i, j) = 2$ for $y_{ij} = 0$. It is updated for exploring the node in each branch-and-bound iteration. With the same example above, the values of $node[]$ for the root node and its two child node are shown in Fig. 3.10.

branch node	$node[]$
root node	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$
$y_{23} = 1$	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$
$y_{23} = 0$	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{pmatrix}$

Figure 3.10: The $node[]$ value for a small example

Afterwards, the following constraints (Eq.3.4.2) and (Eq.3.4.3 need to be added to the subproblem. Note here that, we do not have any additional constraints on the rest of planes of which the corresponding

values of *node* are still 0.

$$(Eq.3.3.14) \quad (Eq.3.3.15) \qquad \forall i, j \in P; node(i, j) = 1 \qquad (3.4.2)$$

$$(Eq.3.3.16) \quad (Eq.3.3.17) \qquad \forall i, j \in P; node(i, j) = 2 \qquad (3.4.3)$$

3.4.3 Branch-and-Bound

The above column generation algorithm provides the lower bound for each node of the branch-and-bound tree. Here, we focus on the branching variables and the node selection. Initially, we pick out the non-zero variables and save these in vector ($Z\_I[]$). The corresponding landing sequence can be found in $Seq[]$. The matrix $Weight[]$ (P by P) represents the connection flow between the planes. It is an implementation of calculating the branching variable y_{ij} by Eq.3.3.13 metioned in section 3.3.2. The two-dimension vector $Branch[]$ is used to save the information of the branch nodes that have been considered. Before we choose a new node, those nodes appearing in $Branch[]$ need to be removed. This can be achieved by resetting the value of these aforementioned node in $Weight[]$ to be 0. Finally, we just pick the largest fractional value in the $Branch[]$ and set the corresponding node to be the next one to be explored. The implementation details are shown in the following figure Fig.3.11. (Codes see Appendix D).

For example, for the same instance as shown in Fig.3.1, we assume that the solution of the linear relaxation of the problem is Z = [0.7 0.2 0.3 0.1 0 0.5 0.2] as shown in Fig.3.12. In the implementation, the matrix $Weight[]$ has the value of

$$Weight = \begin{pmatrix} - & 0.3 & 0 \\ 0 & - & 0.7 \\ 0 & 0 & - \end{pmatrix}$$

Since, the current node is the root node of the branch tree, the $Branch[]$ is empty. The maximum value in the matrix $Weight[]$ is 0.7, hence

```
initialization;
ro = number of nodes in Branch[];
Weight[] = zeros(P,P);
Z_I = find(Z =0);
for i = 1 to length(Z_I)
    seq = Seq(Z_I(i),:)
    for i = 1 to length(seq)-1
        Weight(j,j+1) = Weight(j,j+1) + Z_I(i);
    end
end
for m = 1 to ro
    Weight(Branch(m,1), Branch(m,2))=0
end
max_weight = max(max(Weight));
max_wei_node = find(Weight==max_weight);
return max_wei_node
```

Figure 3.11: The implementation of node selection

$max\_weight = 0.7$. By using the matlab function $find.m$, the index of the max_weight is returned and is stored in $max\_wei\_node[]$ (i.e. max_wei_node = [2,3]). In the next branch-and-bound iteration, the branching node with y_{max_weight}=1 is selected to be explored.

3.4.4 The overall algorithm

The entire algoritm is a combination of the preprocessing, column generation and the branch-and-bound. In our implementation, we start with generating a upper bound UB by the heuristic method $airland\_heuristic()$. During the column generation, the vector $Z[]$ presents the value of the decision variables z in the set partitioning

variable	a'_s	*landing sequence*	*solution* z_s
z_1	[1 0 0]	{1}	0.7
z_2	[0 1 0]	{2}	0.2
z_3	[0 0 1]	{3}	0.3
z_4	[1 1 0]	{1→ 2}	0.1
z_5	[1 0 1]	{1→ 3}	0.0
z_6	[0 1 1]	{2→ 3}	0.5
z_7	[1 1 1]	{1→ 2 → 3}	0.2

Figure 3.12: The example of branch node selection.

model defined in section 3.1. $A[]$, $C[]$, $Seq[]$, $Time[]$ are used to store the information corresponding to the variable $Z[]$, this information includes: binary column, cost, landing sequence and landing time. Function $air1\_Branch(node[])$ is used to excute the column generation given the feasible region defined by $node[]$ using the way mentioned above. The LP solution of the root node is denoted as the LP lower bound of the entire problem LP_LB. The obj_val is the optimal value determined by the column generation for the current branch node. $Next\_node()$ implements the selection of the next node in the branch and bound, and the branch node information $Branch[]$ is uppdated by $Next\_node()$. Boolean variale *delete* shows whether the current node is pruned or not. The implementation is shown in Fig.3.13. The corresponding program is $Algorithm\_air1\_MinSub.m$ in Appendix D.

```
initialization the data parameter;
UB = airland_heuristic();
node = zeros(P,P);
[Z, A, C, I, obj_val, Seq, Time] = air1_Branch(node);
if Z is integer
    Output the optimal solution;
    break
end
ite = 1000;
delete = 0;
for i=1:itr
    if delete = 0
        [node] = Next_node(node)
    else
        update node
    end
    delete = 0
    [Z, A, C, I, obj_val, Seq, Time] = air1_Branch(node);
    if obj_val > UB
        delete = 1;
    elseif Z is integer
        if obj_val = LP_LB
            Output the optimal solution;
            break
        end
        update UB, Best_IP;
        delete = 1;
    end
end
```

Figure 3.13: The implementation of branch-and-price algorithm.

Chapter 4

Case Studies

In this chapter, we start by investigating the effectivity of solving the ALP by the column generation method which previously has not been applied for the problem in the literature. An experiment on a small instance involving 10 aircraft, is first considered, in which the minimun cost, the landing sequence and the landing time corresponding to each column in the entire set partitioning model **SP** are determined before the column generation starts. The computational results show that the column generation method solved the linear relaxation of the **SP** successfully. Then the branch-and-bound is used to determine the optimal solution for the integer problem **SP**. A modified algorithm which is especially efficient for large-scale problems is developed and tested by the public data in OR-Library involving up to 50 aircraft and 4 runways.

4.1 A Small Instance

As we known the branch-and-price method is very problem-dependent, it is more a principle than an algorithm and may be technically difficult to apply to a specific problem in pratice. Practical difficulties may include how to apply the column generation, how to construct the subproblem, and how to choose a branching variable, etc.

In our experiments, we start with a small instance in OR-Library involving 10 aircraft and 2 runways. In the following, we will first investigate the effectivity of the column generation techniques for sloving the **SP**, the linear relaxation problem of the set partitioning model as formulated in section 3.1, in which each column indicates the planes to be landed and the corresponding coefficient in the objective function denotes the cost for landing those planes. It is very important to note that, the column does not state the order of the landing sequence, and the coefficient in the objective function is the minimum cost corresponding to the optimal landings of the planes appeared in the column. Therefore, in each column generation iteration, the information for the corresponding column to be added needs to be specified before entering it into the master problem. In the literature, no method or formulation is developed for generating the columns in the subproblem of the ALP. To get started, in this small instance, we determined all the columns and the corresponding costs, landing sequences and landing time in the preprocessing by using the enumerate method described in section 3.4.

Afterwards, we start column generation for solving the **LSP**. In each iteration, one of the columns which has the minimum reduced cost is then added to the master problem. The reduced costs ($\widetilde{c}_j$) for each column j by the following,

$$\widetilde{c}_j = c_j - \pi a - \lambda < 0 \qquad \forall j \in S$$

where the c_j is the cost for the column a determined in the preprocessing, the π and λ are the dual values in the master problem.

Without loss of generality, the column generation starts with the *dummy* columns and a random feasible solution of landing the first five planes on runway 1 and the other five on runway 2. The computational results are shown in table 4.1. The objective value 'Obj_value' and the minimum reduced cost '$\min \widetilde{c}_j$' for each iteration are reported in this table.

From the row of 'Obj_val', we can see that the objective value was improved by adding the columns with the negative reduced cost through

Iteration	1	2	3	4	5	6
Obj_value	450.00	450.00	450.00	450.00	323.33	323.33
$\min \widetilde{c}_j$	-811	-1201	-2541	-2181	-1194	-1194

Iteration	7	8	9	10	11	12
Obj_value	243.33	209.17	189.00	172.08	148.67	123.33
$\min \widetilde{c}_j$	-554.33	-321.83	-359	-324.33	-289	-169.33

Iteration	13	14	15	16	17
Obj_value	123.33	115.00	90.00	90.00	90.00
$\min \widetilde{c}_j$	-187.67	-216.0	-91.0	-91.0	0

Table 4.1: Column Generation results for the small experiment.

the whole procedure. And the column generation stopped at the 17th iteration, in which the minimum reduced cost was 0 showing that the **LSP** was solved optimally. And it is worth to note that, the optimal solution for **LSP** was 90 which is equal to the optimal solution for this problem presented in Beasley *et al.* (2000). In other words, the lower bound provided by column generation reached the optimal solution of this problem. This small experiment illustrates that the column generation is often not only effective but also very efficient, and provides very good lower bound for solving the ALP.

In the following, the branch-and-bound method is combined to secure an integer solution of the **SP** problem. The column generation method tested above is used to determine the lower bound for each branch node, and the new branching variable proposed in chapter 3.3.2 is used. Only 7 branching decisions were made: $y_{(6,8)} = 1$, $y_{(8,1)} = 1$, $y_{(7,9)} = 1$, $y_{(3,4)} = 1$, $y_{(1,2)} = 1$, $y_{(9,10)} = 1$ and $y_{(4,5)} = 1$. Afterwards, the algorithm achieved the optimal integer solution shown in table 4.2.

runway	landing column	landing sequence	cost
1	[1 1 0 0 0 1 0 1 0 0]	$\{ 6 \rightarrow 8 \rightarrow 1 \rightarrow 2\}$	90
2	[0 0 1 1 1 0 1 0 1 1]	$\{ 3 \rightarrow 4 \rightarrow 5 \rightarrow 7 \rightarrow 9 \rightarrow 10\}$	0
Total Landing Cost			90

Table 4.2: Final results for the small experiment.

As a conclusion for the experiment on this small instance of ALP, the branch-and-price method was demonstrated to be successful for the ALP and very efficient for this instance. Furthermore, this experiment was started with the columns of a random feasible solution. We believe that the algorithm will perform better with a good start point given by a heuristic solution.

4.2 General Problems

As the size of the problem becomes larger, it is computational inefficient to determine the information of all the columns as we have done in the preprocessing of the above small instance. For example, for a problem with 50 planes, there exsit appro. 1.1259×10^{15} feasible columns. How to construct the subproblem for the ALP, how to generate the columns efficiently for large-scale problems are the next things to be investigated. With this in mind, the mathematical model **SUB** for the subproblem and the entire algorithm described in chapter 3 has been developed and tested by the public data in OR-Library involving up to 50 aircraft and 4 runways.

Our first computational experiment (I) consists two parts: In the

first part, we used our branch-and-price algorithm constructed in chapter 3 with the basic initial columns including the columns of the heuristic solution and the *dummy* columns. It is implemented in Matlab 7.0 on a 1000Mhz Pentium PC with 1GB of memmory, and the master problem **LSP** and subroblem **SUB** were solved by GAMS 21.5. The other part of the experiment used the LMIP-based algorithm presented in Beasley *et al*. (2000) (called **LMIP** based exact algorithm) in which the lower bound was provided by **LMIP** and the branch decision was made on the binary variables in the formulation. It is implemented in GAMS.

The computational results are shown in table 4.3. The $'B\&P'$ part are the results of our branch-and-price exact algorithm, while the $'B\&B'$ corresponds to the other algorithm. The first column is the identification of the problem. The following two columns 'P' and 'R' are the number of planes and the number of runways respectively. Each problem has been solved with an increasing number of runways until the optimal solution value dropped to zero (indicating that we have a sufficient number of runways to enable all planes to land on target time). The column 'LSP-IP Gap' represents the gap in percentage between the linear relaxation solution value of column generation (Z_{LSP}) and the optimal integer solution value (Z_{IP}): $100(Z_{IP} - Z_{LSP})/Z_{IP}$. The column 'LMIP-IP Gap' represents the gap in percentage between the linear relaxation solution value of the mixed integer formulation (Z_{LMIP}) and Z_{IP}. The columns named 'No.of Tree Nodes' show the number of tree nodes searched for solving the problem in the branch-and-price algoritm and the **LMIP** based exact algorithm, respectively, while the columns 'Opt. Sol.' represent the final solutions of them. The columns named 'Total Time (sec)' show the running time for the two algorithms. The column 'Tol.Col.' is the total number of columns generated in the branch-and-price algorithm.

It is clear from column 8 and column 11 in table 4.3, that our algorithm successfully find the optimal solution for all the problems.

Hence, our algorithm in practice is an exact algorithm that can solve the ALP optimally in acceptable time.

As it shows in column ′Tot. Col.′, the optimal solutions are achieved by no more than 500 columns generated. For example, for the problem 8 involving 50 aircraft, there exist appro. 1.1259×10^{15} feasible columns in the entire set partitioning problem, however, only 199 columns were used.

In the column ′LSP-IP Gap′, all the values equal to 0, which indicates that for all the problems, the optimal objective values Z_{LSP} of the linear relaxation **LSP** are equal to the corresponding optimal integer solutions Z_{IP} (i.e. $Z_{LSP} = Z_{IP}$). These are excellent lower bounds for the problems. As we have mentioned, a lower bound (Z_{LB}) for the problem can be used to prune the branch node with upper bound less than Z_{LB} during the tree searching. An excellent lower bound might significantly prune the tree and enhance the efficiency of the branch-and-bound. In column ′LMIP-IP Gap′, the optimality gaps are 100% given by the **LMIP**. In other words, for all the problems, the lower bound Z_{LMIP} equal to 0. This lower bound is trival since it is obvious that cost will never be negative. By comparing these two columns, we reach the conclusion that the lower bound provided by the **LSP** is significantly better than **MIP**.

In the columns of ′No. of Tree Node′, less than 12 tree nodes were explored to find the optimal solution in our branch-and-price algorithm, while extreme large number of nodes in the LP-based tree search algorithm, such as 282160 for problem 5 with 2 runways. This demonstrates that our branch decision makes branch-and-bound much more efficient than the LP-based tree search, in which branching decisions are made on the binary variables in the mixed integer formulation. Moreover, in the problem 1 2 3 4 and 8, only one tree node was explored. In other words, the branch-and-bound procedure stopped at its root node. This is because for these problems, the solutions of the **LSP** provided by the column generation are integer solutions which are also feasible and

optimal to the integer problem **SP**. This once again shows the effectiveness of applying the set partitioning model and column generation on the aircraft landing problems.

However, comparing the running time in column 7 with that in column 11, our branch-and-price algorithm is currently not competitive. Most of the time is used for solving the subproblems in our algorithm. Besides, our algorithm is implemented in Matlab which is not as efficient as C++ or JAVA. In the future, improvements can be made in order to enhance the efficiency of the branch-and-price algorithm. One way is to use a fast heuristic method instead of solving **SUB** at the beginning of the column generation procedure. Another way, which is investigated in our next computational experiment, is to generate some good initial columns for the column generation.

As a conclusion of the comparision between the two exact algorithms, the advantages of our branch-and-price exact algorithm include the following: Our set partitioning fomulation **SP** is a much better formulation of ALP than the mixed integer formulaiton **MIP**, its linear relaxation provides a lower bound that is significantly better than the **MIP**. The branch-and-bound strategies used in our algorithm is much more efficient than the tree search in the LP-based exact algorithm according to the number of the branch nodes explored. This indicates that the proposed branching decision method is very suitable and good for aircraft landing problems. Furthermore, the column generation is an efficient way to solve the **LSP** due to the low number of columns generated for solving the entire problem. These points show the potential of solving very large aircraft landing problems optimally by the branch-and-price algorithm, while the branch-and-bound algorithm presumably is too inefficient and may lead to an excessive computational time in search of the optimum.

Our next computational experiment (II) aims at invesitigating the initial columns. As discussed in chapter 3.4.2, starting with some *good*

additional columns may shorten the computational time. In the following, we added the additional columns constructed in the way described in section 3.4.2. The computational results are shown in table 4.4.

For this experiment, the general conclusions which are the same as we have got in experiment I, will not be discussed in details, including: the **LSP** provide good lower bounds for the problem, the algorithm achieve optimal, only few branch nodes had been explored and quite few columns were generated. Here, we focus on the different performances between experiment I and II in order to investigate the efficiency of adding additional columns at the beginning.

In table 4.4, the new column named ′Add.Col.′ shows the number of additional columns generated for each problem. We can observe that the larger the problem is the more additional columns were generated in general, such as for the problem 8 involving 50 aircraft, 18 additional columns were generated by our generating strategy stated in section 3.4.2.

Comparing the total running time of these two experiments (column 9 and column 11 in table 4.4), all the problem were solved faster in II than I except two of them, the problem 4 with 2 runways and the problem 5. In particular, for the first three problems, the addition of these kind of columns significantly shortened the running time. This indicates that choosing proper initial columns may improve the efficiency of the algorithm.

If we compare the total number of columns used (column 7 and column 10 in table 4.4), most of the problems in I used less column to reach optimality. For problem 8, 158 columns used in II while 199 in I – 25% columns are saved by the additional initial columns.

As a conclusion of the comparison between I and II, the selection of the initial columns in column generation has effect on the efficiency of solving the ALP. More investigation and experiments must however be made on the initial columns in order to make the branch-and-price algorithm more efficient.

			$B\&P$					$B\&B$			
Pro. Num.	P	R	LSP -IP Gap	No.of Tree Nodes	Tot. Col.	Total Time (sec)	Opt. Sol.	LMIP -IP Gap	No.of Tree Nodes	Total Time (sec)	Opt. Sol.
1	10	2	0.0	1	25	36.4	90	100	91	0.13	90
		3	-	-	-	-	0	-	-	-	0
2	15	2	0.0	1	43	72.8	210	100	115	0.13	210
		3	-	-	-	-	0	-	-	-	0
3	20	2	0.0	1	56	96.4	60	100	142	0.27	60
		3	-	-	-	-	0	-	-	-	0
4	20	2	0.0	1	93	267.9	640	100	193319	417.2	640
		3	0.0	1	61	117.0	130	100	39901	7.48	130
		4	-	-	-	-	0	-	-	-	0
5	20	2	0.0	9	192	1133.3	650	100	282160	130.74	650
		3	0.0	12	142	624.2	170	100	20035	21.30	170
		4	-	-	-	-	0	-	-	-	0
6	30	2	0.0	9	461	2960.6	554	100	25316	5.86	554
		3	-	-	-	-	0	-	-	-	0
7	44	2	-	-	-	-	0	-	-	-	0
8	50	2	0.0	1	199	769.7	135	100	9020	4.56	135
		3	-	-	-	-	0	-	-	-	0

Table 4.3: Computational results for experiment I

			II						I	
Pro. Num.	P	R	LSP -IP Gap	No.of Tree Nodes	Add. Col.	Tot. Col.	Total Time (sec)	Opt. Sol.	Tot. Col.	Total Time (sec)
1	10	2	0.0	1	4	18	7.5	90	25	36.4
		3	-	-	-	-	-	0	-	-
2	15	2	0.0	1	6	26	10.0	210	43	72.8
		3	-	-	-	-	-	0	-	-
3	20	2	0.0	1	6	33	15.5	60	56	96.4
		3	-	-	-	-	-	0	-	-
4	20	2	0.0	1	10	86	304.1	640	93	267.9
		3	0.0	1	10	53	59.7	130	93	117.0
		4	-	-	-	-	-	0	-	-
5	20	2	0.0	9	9	204	1626.6	650	192	1133.3
		3	0.0	12	9	98	790.8	170	142	624.2
		4	-	-	-	-	-	0	-	-
6	30	2	0.0	9	16	446	2854.1	554	461	2960.6
		3	-	-	-	-	-	0	-	-
7	44	2	-	-	-	-	-	0	-	-
8	50	2	0.0	1	18	158	502.9	135	199	769.7
		3	-	-	-	-	-	0	-	-

Table 4.4: Computational results for experiment II

Chapter 5

Conclusion

In this chapter, we summarize the achievements in this thesis and point out some future research in the field of aircraft landing problem.

5.1 Achievements

This thesis is the first attempt to develop and implement a branch-and-price algorithm for the aircraft landing problem. The achievements of this thesis are shown in the following aspects:

- We reformulated the ALP as a set partitioning problem with side constraints on the limited number of available runways. We then presented a set partitioning model (**SP**) for the problem.
- We adopted the column generation method for solving the linear relaxation of the set partitioning model (**LSP**) to determine a lower bound of the optimal integer solution. Computational results show that our **LSP** provide a quite good lower bound while the linear relaxation of the mixed integer formulation (**LMIP**) can not achieve. For each of the problems we have tested, the gap between the solution of **LSP** and the optimal solution is 0,

indicating our **SP** formulation is much better than the **LMIP** formulation for the aircraft landing problem.

- We presented a mixed integer formulation **SUB** for the subproblem involved in the column generation, which is used to generate the column with the minimum reduced cost in each column generation iteration. Moreover, the corresponding total cost, landing sequence and landing time for each plane appearing in the column can also be determined by solving the **SUB**. In order to make it available for solving the node in branch-and-bound tree, we proposed 4 constraints ensuring that the column we generate is in the feasible region of the node.

- We extended the column generation into a branch-and-price framework, in which the branch-and-bound method is used to guarantee that we end up with an integer solution. In other words, we proposed an exact algorithm based on the branch-and-price principle to solve the optimal solution of the ALP. We introduced a new branching variable which is determined by the total flow connecting the successive planes in the fractional solution. The computational results show that for all the problems, the algorithm achieves optimal solution with no more than 12 branch-and-bound nodes explored, indicating the branch decision strategy proposed in this algorithm makes the branch-and-bound quite efficient for the ALP.

- We implemented the exact algorithm and tested it using the public data from OR-Library involving up to 50 aircraft and 4 runways. Furthermore, it is worthwhile to note that all of our problems were solved with less than 450 columns generated and 12 branch nodes explored. This indicates potential capability of solving large-scale aircraft landing problems optimally by the branch-and-price algorithm.

5.2 Future research

As a result of the work presented in this thesis, the following future research on the ALP are suggested:

- Development of a strategy for generating $'good'$ initial columns

  The experiments presented in this thesis show that the running time can be shortened by adding additional proper initial columns at the beginning of the column generation. In our computational experiments, this is achieved by adding the set of initial columns generated by the strategy given in section 3.4.2 for most of the problems. Further work can be conducted to investigate which initial columns will be more probable to be helpful to the problems in general, how many we should generate, and also a strategy constructed to generate the initial columns.

- Development of a heuristic method for the subproblem

  In the column generation method, it is necessary to solve the optimal solution of the mixed integer model of the subproblem **SUB** in order to ensure the optimality of the master problem (e.g. $\min \tilde{c}_j \geq 0$). However, at the beginning of the column generation, especially in the case that the current solution is far from the optimal LP solution, there might exist many columns with negative reduced cost that can be added in order to improve the current solution. It is time consuming to solve the subproblem by the method proposed in this thesis. In order to speed up, a fast heuristic method may be developed and used for generating the columns to be added at the beninning of the column generation.

  Note that the ALP can be viewed as a Vehicle Routing Problem with Time Windows (VRPTW) with a target service (landing) time, an infinite capacity, and an objective function of minimizing the total deviation from the taget time. The subproblem of

the VRPTW has been successfully solved by Elementary Shortest Path Problem with Time Windows and Capacity Consitraints (ESPPTWCC) (see Larsen (1999)). In the future, we plan to develop a heuristic method based on the method of solving the Elementary Shortest Path Problem for the subproblem of the ALP.

- Investigation on the cuts (valid inequalities) for the problem

  In our algorithm, the method used to guarantee the integrality of the solution after solving the linear relaxation of the original problem is branch-and-bound method. Another method is to improve the polyhedal description of the relaxed problem in order to get an integer solution or at least tighten the bound by adding cuts (inequalities). As we mentioned in chapter 3, the subproblem is similar to a single runway landing problem, which can be viewed as an open Travelling Salesman Problem (TSP) (Bianco *et al.* (1993)). For the TSP, several kinds of cuts has been successfully applied: the comb-inequlities, 2-path inequalities etc (Bard *et al.* (2002)). In the future, investigation can be made in searching for cuts for the ALP that can be added into the LP relaxation formulation and which results in an improved optimal solution by solving the new enhanced problem. Furthmore, the cuts can be generated and added throughout the whole branch-and-price algorithm resulting in a branch-and-price-and-cut algorithm for the ALP.

- Investigation on the dynamic case of the ALP

  The research in this thesis focusses on the static case of the ALP. This is particularly useful to investigate airport runway capacity in the strategic planning stage. However, during the airport operation, the information of the landings, such as the earlist, target and latest time, might change over time. Therefore, the dynamic ALP is crucial to the applicability in operational planning.

Bibliography

[1] J. E. Beasley, M. Krishnamoorthy, Y. M. Sharaiha, and D. Abramson, Scheduling Aircraft Landings-the static case, *Transportation Science*, 34(2), 180-197, 2000.

[2] A. T. Ernst, M. Krishnamoorthy, and T. H. Storer, Heuristic and Exact Algorithms for Scheduling Aircraft Landings, *Networks*, 34, 229-241, 1999.

[3] H. Pinol, and J. E. Beasley, Scatter Search and Bionomic Algorithms for the Aircraft Landing Problem, to appear in the *European Journal of Operational Research*, Currently available from http://people.brunel.ac.uk/ mastjjb/jeb/jeb.html, 2004.

[4] G. Jung, and M. Laguna, Time segmenting heuristic for an aircraft landing problem, Working paper, Leeds School of Business, University of Colorado, Boulder, CO 80309-0419, USA. Submitted for publication. Currently available from http://leeds-faculty.colorado.edu/laguna/articles/tsh.html, March, 2003 .

[5] J. Abela, D. Abramson, M. Krishnamoorthy, A. De Silva, and G. Mills, Computing optimal schedules for landing aircraft, *Proceedings of the 12th National ASOR Conference*, 71-90, 1993.

[6] J. E. Beasley, J. Sonander, and P. Havelock, Scheduling aircraft landings at London Heathrow using a population heuristic, *Journal of the Operational Research Society*, 52, 483-493, 2001.

[7] V. Ciesielski, and P. Scerri, Real time genetic scheduling of aircraft landing times, *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation (ICEC98)*, 360-364, 1998.

[8] V. H. L. Cheng, L. S. Crawford, and P. K. Menon, Air traffic control using genetic search techniques, *International Conference on Control Application*, 1999.

[9] L. Bianco, A. Mingozzi and A. Ricciardelli, The Traveling Salesman Problem with Cumulative Costs, *Networks*, 23, 81-91, 1993.

[10] M. J. Soomer, G. J. Franx, Scheduling Aircraft Landing using Airlines' Preferences, http://www.math.vu.nl/ mjsoomer/aircraftlandings.pdf, Aug, 2005.

[11] L. Bianco and M. Bielli, System Aspects and Optimization Models in ATC Planning, *Large Scale Computation and Information Processing in Air Traffic Control*, 47-99, 1993.

[12] L. Bianco, P. Dellolmo and S. Giordani, Minimizing Total Completion Time Subject to Release Dates and Sequence Dependent Processing Times, *Annals of Operations Research*, 86, 393-415, 1999.

[13] H. N. Psaraftis, A Dynamic Programming Approach to the Aircraft Sequencing Problem, Report R78-4, Flight Transportation Laboratory, MIT, Cambridge, MA, 1978.

[14] H. N. Psaraftis, A Dynamic Programming Approach Sequencing Groups of Identical Jobs, *Opns. Res.*, 28, 1347-1359, 1980.

[15] J. E. Beasley, M. Krishnamoorthy, Y. M. Sharaiha, and D. Abranmson, Displacement problem and dynamically scheduling aircraft landings, *Journal of the Operational Research Society*, 55:54-64, 2004.

[16] J. A. D. Atkin, E. K. Burke, J. S. Greenwood, and D. Reeson, A meta-heuritstic approach to aircraft departure scheduling at London Heathrow airport, Currently available at: http://fugazi.engr.arizona.edu/caspt/atkin.pdf, 2005.

[17] J. Larsen, Parallelization of the Vehicle Routing Problem with Time Windows, *PhD thesis, IMM, DTU*, 1999.

[18] Z. L. Chen and W. B. Powell, A Column Generation Based Decomposition Algorithm for a Parallel Machine Just-In-Time Scheduling Problem, *European Journal of Operational Research*, 116, 220-232, 1997.

[19] M. E. Lubbecke, and J. Desrosiers, Selected Topic in Column Geceration, Technical Report G-2002-64, GERAD, Montreal, December 2002.

[20] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance, Branch-and-price: Column generation for solving huge integer programs, *Operational Research*, 46(3): 316-329, 1998.

[21] G. Desaulniers, J. Desrosiers, Y. Dumas, M. M. Solomon, Accecerating strategies in column generation methods for vehicle routing and crew scheduling problems, *Essays and surveys in metaheuristics*, 309-324, 2001.

[22] D. M. Ryan and B. A. Foster, An integer programming approach to scheduling, *Computer Scheduling of Public Transport*, 269-280, 1981.

[23] J. C. Mello, Air Transport, http://www.mre.gov.br/cdbrasil/itamaraty/web/ingles/economia/transp/aereo/apresent.htm,2002.

Appendix A

Matlab programs for heuristic mehtod

```
function [C,t,X_star,x,Seq,Xrun,C_seq,E,T,L,g,h,S]
         = airland_heuristic(Pro,Run)
% Find the upbound (a feasible solution) by using the heuristic method
%
% Usage:    [C,A,x,E,T,L,g,h,S] = airland(Pro,Run)
%
% Input:    Pro:    The number of the problem
%           Run:    The number of the runways
% Output:   C:      The total cost
%           x:      Landing time for each plane
%           A:      The order of the planes on each runway'
%           t:      Time consuming
%           E:      The earliest landing time
%           T:      The target landing time
%           L:      The latest landing time
%           g:      The unit cost of landing before target time
%           h:      The unit cost of landing after target time
%           S:      The seperation time between the planes
% starting time
tic
% Get the data
[P,E,T,L,g,h,S] = Getdata(Pro);
% Get the index corresponding to the sorted plane
ord = sortindex(T);
% initialization
mS = zeros(1,P);
A = zeros(Run,P+1);
x = zeros(1,P);
X_star = zeros(1,P);
C = 0;
```

```
Xrun = zeros(Run,P);
if Run == 1
    C = 1;
    A = ord;
end
if Run > 1
    % for each planes
    for i = 1:P
        j = ord(i);
        B = zeros(1,Run);
        % at each runway
        for r = 1:Run
            mS = zeros(1,P);
            l = 1;
            if (find(A(r,:)>0))
                Q = find(A(r,:)>0);
                for m = 1:length(Q)
                    k = A(r, Q(m));
                    mS(l) = x(k) + S(k,j);
                    l = l+1;
                end
            end
            B(r) = max (T(j), max(mS));
        end
        % the best landing time for plane j
        x(j) = min(B);
        % feasible runway for x(j)
        rway = find(B==min(B));
        emp = find (A(rway(1),:) == 0);
        % put j into the runway r
        A(rway(1), emp(1)) = j;

        % add the cost of the current plane to the total cost
        if x(j)<T(j)
            C = C+(T(j)-x(j))*g(j);
        else
            C = C+(x(j)-T(j))*h(j);
        end
    end
end
C_seq=zeros(Run,1);
% If C>0, recalculate the solution
if C~=0
    C = 0;
    Q =[];
    % for each runway
    for r = 1:Run
        n =length(find(A(r,:)>0));
        Q = A(r,1:n);
        [f,AA,b,Aeq,beq,lb,ub] = recalculateX(Q,g,h,T,E,L,S);
```

```
        [xcurrent,fval] = linprog(f,AA,b,Aeq,beq,lb,ub);
        Xrun(r,1:n) = xcurrent(2*n+1:end);
        X_star = opttime(X_star,Xrun(r,:),Q);
        [B,false,SS,xx]=seperationtest(X_star,S,Q);
        if false == 1
            [f,AA,b,Aeq,beq,lb,ub] = recalculateXXXX(Q);
            [xcurrent,fval] = linprog(f,AA,b,Aeq,beq,lb,ub);
            Xrun(r,1:n) = xcurrent(2*n+1:end);
            X_star = opttime(X_star,Xrun(r,:),Q);
            Rere = 1
        end
        % calculating the cost
        C_seq(r)=fval;
        C = C + fval;
        clear xcurrent;
    end
end
% Output the sequence and cost
C;
Seq = A ;
t=toc ;

function v = sortindex(a)
% Get the index corresponding to the sorted plane
%
% Usage:    [v , c]= sortindex(a)
%
%Input:     a:  The vector which want to be sorted
%Output:    v:  The vector of the original index
b = sort(a);
for i = 1:length(a)
    c = find(a==b(i));
    d = c(1);
    a(d) = NaN;
    v(i) = c(1);
end

function [xx,gg,hh,TT,EE,LL,SS] = getxx(A,x,r,g,h,T,E,L,S)
% Get the properties of the planes which will land on runway r
% with the fix order in the Step 1 and 2
% current planes on runway r
Q = find(A(r,:)>0);
for m = 1:length(Q)
    k = A(r, m);
    k1 = A(r, m+1);
    xx(m) = x(k);
    gg(m) = g(k);
    hh(m) = h(k);
    TT(m) = T(k);
    EE(m) = E(k);
```

```
    LL(m) = L(k);
    if m < length(Q)
        SS(m) = S(k, k1);
    end
end

function [f,A,b,Aeq,beq,lb,ub] = recalculateX(A,x,r,g,h,T,E,L,S)
% Form the model to be minimizing as following
%       Min     f'x
%       s.t.    A*x <= b
%               Aeq*x = beq
%               lb <= x <= ub
%
% Usage:    [f,A,b,Aeq,beq,lb,ub] = recalculateX(A,x,r,g,h,T,E,L,S)%
% Input:    A:      The order of the planes on each runway
%           x:      current solution
%           r:      the runway to be recalculated
%           g:      The unit cost of landing before target time
%           h:      The unit cost of landing after target time
%           T:      The target landing time
%           E:      The earliest landing time
%           L:      The latest landing time
%           S:      The seperation time between the planes
% Output:   the parameters in the model
%           f:      The coefficient for variable
%                   [a1 a2 ... b1 b2 ... x1 x2 ...]
%           A:      The coefficient matrix of the left handside of
%                   x(i)-x(i+1) <= S(i,i+1)
%           b:      The vector of right handside of
%                   x(i)-x(i+1) <= S(i,i+1)
%           Aeq:    The leftside of a(i)-b(i)+x(i) == T(i)
%           beq:    The rightside of a(i)-b(i)+x(i) == T(i)
%           lb:     The lowerbound of the variable
%           ub:     The upperbound of the variable
% Get the properties of the planes which will land on runway r with
% the fix order in the Step 1 and 2
[xx,gg,hh,TT,EE,LL,SS] = getxx(A,x,r,g,h,T,E,L,S);
n = length(gg);
ze = zeros(n,1);
z = zeros(n-1,n);
I = eye(n);
f = [gg';hh';ze];
  sum gg(i)*a(i)+hh(i)*b(i)+0*x(i)
A = [z z eye(n-1) zeros(n-1,1)] + [z z zeros(n-1,1) -eye(n-1)];
b = -SS';
lb = [ze;ze;EE'];
ub = [TT'-EE';LL'-TT';LL'];
beq = [TT'];
Aeq = [I -I I];
```

```
function [P,E,T,L,g,h,S] = Getdata(Pro)
% Find the upbound (a feasible solution) by using the heuristic method
%
% Usage:    [C,A,x,E,T,L,g,h,S] = airland(Pro,Run)
%
% Input:    Pro:    The number of the problem
%           Run:    The number of the runways
% Output:   C:      The total cost
%           x:      Landing time for each plane
%           A:      The order of the planes on each runway'
%           t:      Time consuming
%           E:      The earliest landing time
%           T:      The target landing time
%           L:      The latest landing time
%           g:      The unit cost of landing before target time
%           h:      The unit cost of landing after target time
%           S:      The seperation time between the planes
% Read the data from 'airland .txt'
switch Pro,
    case 1,
        Dat = textread('airland1.txt');
    case 2,
        Dat = textread('airland2.txt');
    case 3,
        Dat = textread('airland3.txt');
    case 4,
        Dat = textread('airland4.txt');
    case 5,
        Dat = textread('airland5.txt');
    case 6,
        Dat = textread('airland6.txt');
    case 7,
        Dat = textread('airland7.txt');
    case 8,
        Dat = textread('airland8.txt');
    case 9,
        Dat = textread('airland9.txt');
    case 10,
        Dat = textread('airland10.txt');
    case 11,
        Dat = textread('airland11.txt');
    case 12,
        Dat = textread('airland12.txt');
    case 13,
        Dat = textread('airland13.txt');
end

% Get the data
if Pro < 2
    d1 = Dat(2:3:end, 1:end-3);
```

```
    S = [Dat(3:3:end, 1:end-1) Dat(4:3:end, 1:2)];
elseif Pro < 3
    d1 = Dat(2:3:end, 1:end-3);
    S = [Dat(3:3:end, 1:end-1) Dat(4:3:end, 1:end-2)];
elseif (Pro < 6)
    d1 = Dat(2:4:end, 1:end-3);
    S = [Dat(3:4:end, 1:end-1) Dat(4:4:end, 1:end-1) ...
         Dat(5:4:end, 1:end-5) ];
elseif (Pro == 6)
    d1 = Dat(2:5:end, 1:end-3);
    S = [Dat(3:5:end, 1:end-1) Dat(4:5:end, 1:end-1) ...
         Dat(5:5:end, 1:end-1) Dat(6:5:end, 1:end-3) ];
elseif (Pro == 7)
    d1 = Dat(2:7:end, 1:end-3);
    S = [Dat(3:7:end, 1:end-1) Dat(4:7:end, 1:end-1) ...
         Dat(5:7:end, 1:end-1) Dat(6:7:end, 1:end-1) ...
         Dat(7:7:end, 1:end-1) Dat(8:7:end, 1:end-5) ];
elseif (Pro == 8)
    d1 = Dat(2:8:end, 1:end-3);
    S = [Dat(3:8:end, 1:end-1) Dat(4:8:end, 1:end-1) ...
         Dat(5:8:end, 1:end-1) Dat(6:8:end, 1:end-1) ...
         Dat(7:8:end, 1:end-1) Dat(8:8:end, 1:end-1) ...
         Dat(9:8:end, 1:end-7) ];
elseif (Pro == 9)
    d1 = Dat(2:5:end, 1:6);
    S = [Dat(3:5:end, 1:end) Dat(4:5:end, 1:end) ...
         Dat(5:5:end, 1:end) Dat(6:5:end, 1:10) ];
elseif (Pro == 10)
    d1 = Dat(2:6:end, 1:6);
    S = [Dat(3:6:end, 1:end) Dat(4:6:end, 1:end) ...
         Dat(5:6:end, 1:end) Dat(6:6:end, 1:end) ...
         Dat(7:6:end, 1:end) ];
elseif (Pro == 11)
    d1 = Dat(2:8:end, 1:6);
    S = [Dat(3:8:end, 1:end) Dat(4:8:end, 1:end) ...
         Dat(5:8:end, 1:end) Dat(6:8:end, 1:end) ...
         Dat(7:8:end, 1:end) Dat(8:8:end, 1:end) ...
         Dat(9:8:end, 1:20) ];
elseif (Pro == 12)
    d1 = Dat(2:10:end, 1:6);
    S = [Dat(3:10:end, 1:end) Dat(4:10:end, 1:end) ...
         Dat(5:10:end, 1:end) Dat(6:10:end, 1:end) ...
         Dat(7:10:end, 1:end) Dat(8:10:end, 1:end) ...
         Dat(9:10:end, 1:end) Dat(10:10:end, 1:end) ...
         Dat(11:10:end, 1:10) ];
else
    d1 = Dat(2:18:end, 1:6);
    S = [Dat(3:18:end, 1:end) Dat(4:18:end, 1:end) ...
         Dat(5:18:end, 1:end) Dat(6:18:end, 1:end) ...
         Dat(7:18:end, 1:end) Dat(8:18:end, 1:end) ...
```

```
        Dat(9:18:end, 1:end) Dat(10:18:end, 1:end) ...
        Dat(11:18:end, 1:end) Dat(12:18:end, 1:end) ...
        Dat(13:18:end, 1:end) Dat(14:18:end, 1:end) ...
        Dat(15:18:end, 1:end) Dat(16:18:end, 1:end) ...
        Dat(17:18:end, 1:end) Dat(18:18:end, 1:end) ...
        Dat(19:18:end, 1:20)];
end
% Sort the data
P = Dat(1,1);
E = d1(:, 2);   T = d1(:, 3);   L = d1(:, 4);
g = d1(:, 5);   h = d1(:, 6);
```

Appendix B

Matlab programs for Column Generation

```
function [Z,A,C,obj_val,Seq,Time] = air1_Branch(node_1,node_0)

% Solve the linear relaxation by column generation
% given the previous information and the branching condition
%
% Usage:  [Z,A,C,obj_val,Seq,Time] = air1_Branch(node_1,node_0)
%
% Input:  prev_I:  The final i(s) corresponding to the
%                  final columns in the previous iteration
%         avail_I: The available i(s) in the previous iteration
%                  (new available i(s) are found within this set )
% Output: Z:       The optimal solution for this branch
%         A:       The final columns in this iteration
%         C:       The final objective function coefficient
%         I:       The final i(s) corresponding to the final columns
%           obj_val: The objectiv function in each iteration
global P R E T L g h S oriE oriL
global A C Seq Time
global TT1 TT2 TT3 TT4 TT5 TT6
B = ones(P,1);
% Find out the infeasible columns, set Fix to be 1
Fix= checkavail(node_1,node_0);
% Update the node[] for the current node
node_parameter = zeros(P,P);
[ro_1,co_1] = size(node_1);
[ro_0,co_0] = size(node_0);
if ~isempty(node_1)
    for i = 1:ro_1
        node_parameter(node_1(i,1),node_1(i,2))=1;
        % the other elements in the same row
```

```
        node_parameter(node_1(i,1),1:node_1(i,2)-1) = 2;
        node_parameter(node_1(i,1),node_1(i,2)+1:end)=2;
        node_parameter(node_1(i,1),node_1(i,1)) = 0;
        % the other elements in the same column
        node_parameter(1:(node_1(i,1)-1),node_1(i,2)) = 2;
        node_parameter(node_1(i,1)+1:end,node_1(i,2))=2;
        node_parameter(node_1(i,2),node_1(i,2)) = 0;
    end
end
if ~isempty(node_0)
    for i = 1:ro_0
        node_parameter(node_0(i,1),node_0(i,2))=2;
    end
end
node = node_parameter;
% The number of the iterations
Num = 1000000;
obj_val =[];
temp = ones(P,1);
Fix(1:P)=zeros(P,1);
for n = 1:Num
    % Solve the master problem by 'air.gms'
[z,u,ulast,obj] = gams('air',A,B,C,temp,Fix,R);
master_solution = find(z.val~=0);
    size(A);
   % The objective value
obj_val = [obj_val z.val(1:length(C))'*C];
   % The dual variables for the equations
pi = u.val;
    dual_value = pi';
pilast = ulast.val;
% Solve the subproblem by 'subproblem.gms'
[x,col,Re_cost] = gams('subproblem',E,T,L,g,h,S,pi,pilast,node);
    if Re_cost.val > -0.000001
        Z = z.val(1:length(C))';
       break
    end
    for i = P+1:(length(C))
    if A(:,i)==col.val
            here=1;
        end
    end
    c= Re_cost.val + [pi' pilast]*[col.val; 1];
    temp_seq = sortindex(x.val);
    s = [temp_seq(length(find(x.val==0))+1:end) ...
         zeros(1,P-length(find(x.val~=0)))];
    ti = [sort(x.val(find(x.val~=0)))
          zeros(P-length(find(x.val~=0)),1)]';
    % Update the C[], A[], Seq[], Time[], and Fix[]
    C = [C; c];
```

```
    A = [A col.val ];
    Seq = [Seq; s];
    Time = [Time; ti];
    Fix(length(C))=0;
end

function   initialization(p,r)
% initialize the parameters P, E[], T[], g[], h[]
% initialize matrix for saving the computational results for
% A[], C[], I[], Seq[], Time[]
% initialize the initial columns for column generation
global P E T L g h S R oriE oriL
global A C I Seq Time
global UB All_Nodes Nnode
%parameter initialization
Pro = p;
R = r;
[P,oriE,T,oriL,g,h,S] = Getdata(Pro);
for i = 1:P
    S(i,i) = 9999;
end
% Calculate the upper bound by airland_heuristic.m
[UB,t,X_star,x,heuristic_Seq,heuristic_Time,heuristic_C]
= airland_heuristic(p,r);
All_Nodes = avail_nodes_90;
Nnode = length(All_Nodes);
ordind = sortindex(T);
for i = 1:P
    E(i) = max(oriE(i), T(i)-UB/g(i));
    L(i) = min(oriL(i), T(i)+UB/h(i));
end
% the 'dummy' part of the initial column
C = ones(P,1)*5000;
Seq = [[1:P]', zeros(P,P-1)];
A = eye(P);
Time = [T,zeros(P,P-1) ] ;
% the 'heuristic solution' part of the initial column
heuristic_col = zeros(R,P);
for i = 1:R
    for j = 1:length(find(heuristic_Seq(i,:)~=0))
        heuristic_col(i,heuristic_Seq(i,j))=1;
    end
end
C = [C; heuristic_C];
Seq = [Seq; heuristic_Seq(:,1:P)];
Time = [Time; heuristic_Time];
A = [A, heuristic_col'];
% the good initial columns generated by ini_good_col.m
[cost0_c,cost0_col,cost0_Seq,cost0_Time] = ini_good_cols(Pro);
C = [C; cost0_c];
```

```
Seq = [Seq; cost0_Seq];
Time = [Time; cost0_Time];
A = [A, cost0_col'];

function Fix = checkavail(node_yes,node_no)
global Seq
% determine the Fix value for those sequences in Seq
[ro,co] = size(Seq);
Fix = zeros(10000,1);
[ro_yes,co_no] = size(node_yes);
[ro_no,co_no] = size(node_no);
for i = 1:ro
    Seq_i = Seq(i,:);
    for y = 1:ro_yes
        index_1 = find(Seq_i==node_yes(y,1));
        index_2 = find(Seq_i==node_yes(y,2));
        if isempty(index_1) & isempty(index_2)
        elseif index_1 & index_2 & (index_1 == index_2 - 1)
        else
            Fix(i) = 1;
            break
        end
    end
    for n = 1:ro_no
        index_1 = find(Seq_i==node_no(n,1));
        index_2 = find(Seq_i==node_no(n,2));
        if isempty(index_1) & isempty(index_2)
        elseif index_1 & index_2 & (index_1 == index_2 - 1)
            Fix(i) = 1;
            break
        end
    end
end
Fix(ro+1:end) = 1;

function [cost0_col,cost0_Seq,cost0_Time] = ini_good_cols(Pro)
% Find the 'good' initial columns
global P E T L g h S
% [P,E,T,L,g,h,S] = Getdata(Pro);
num = 50;
no = round(P*0.85);
cost0_c = zeros(num,1);
cost0_col = zeros(num, P);
cost0_Seq = [];
cost0_Time = [];
nonexist = [];
ordindT = sortindex(T);
TT = sort(T);
t = []; s = [];
cost0_col(1,ordindT(1)) = 1;
```

```
t = TT(1);
s = ordindT(1);
for j = 2:P
    q = find(cost0_col(1,:)~=0);
    if max(T(q) + S(q,ordindT(j))) <= TT(j)
        cost0_col(1,ordindT(j)) = 1;
        t = [t TT(j)];
        s = [s ordindT(j)];
    else
        nonexist = [nonexist j];
    end
end
cost0_Time = [cost0_Time; t, zeros(1, P-length(t))];
cost0_Seq = [cost0_Seq; s, zeros(1, P-length(s))];
for i = 1:length(nonexist)
    t = []; s = [];
    for j = 1:P
        if j < nonexist(i)
            q = find(cost0_col(i+1,ordindT(1:nonexist(i))));
            if (TT(j)+S(j, ordindT(j))) <= TT(nonexist(i))
                if isempty(q)
                    cost0_col(i+1,ordindT(j)) = 1;
                    t = [t TT(j)];
                    s = [s ordindT(j)];
                elseif max(TT(q) + S(ordindT(q),ordindT(j))) <= TT(j)
                    cost0_col(i+1,ordindT(j)) = 1;
                    t = [t TT(j)];
                    s = [s ordindT(j)];
                end
            end
        elseif j == nonexist(i)
                cost0_col(i+1,ordindT(j)) = 1;
                t = [t TT(j)];
                s = [s ordindT(j)];
        else
            q = find(cost0_col(i+1,:));
            if max(T(q) + S(q,ordindT(j))) <= TT(j)
                cost0_col(i+1,ordindT(j)) = 1;
                t = [t TT(j)];
                s = [s ordindT(j)];
            end
        end
    end
    cost0_Time = [cost0_Time; t, zeros(1, P-length(t))];
    cost0_Seq = [cost0_Seq; s, zeros(1, P-length(s))];
end
for i = 1:num
    if cost0_col(i,:)==0
        cost0_c = cost0_c(1:i-1);
        cost0_col = cost0_col(1:i-1,:);
```

```
        cost0_Seq = cost0_Seq(1:i-1,:);
        cost0_Time = cost0_Time(1:i-1,:);
        break
    end
end
```

Appendix C

GAMS programs for the master problem and the subproblem

```
Air1.gms

Set
i /1*100/
j /1*10000/;
Parameter
A(i,j)          the column matrix
B(i)            each plane lands on one runway
C(j)            the cost coefficient
Fix(j)          the control variable for z
temp(j)         the column using runway;
Scalars
R       the number of runways  /2 /;
Variable
zzz     the sum of the cost;
Positive variable
z(j)    the decision variable;
Equation
cost    the objective function
dual(i) the row for each plane
duallast        the limit on the number of available runways
check(j)        the feasibility of the decision variable;
cost..          zzz =e= sum(j, C(j)*z(j)) ;
dual(i)..       sum(j, A(i,j)*z(j)) =e= B(i);
duallast..      sum(j$(temp(j)=0), z(j))=e= R;
check(j)$(Fix(j) > 0) ..z(j) =e= 0;
model air1 /all/;
```

```
$if exist matdata.gms $include matdata.gms
solve air1 using lp minimizing zzz;
display zzz.l;
$libinclude matout z.l J
$libinclude matout dual.m I
$libinclude matout duallast.m
$libinclude matout zzz.l


Subproblem.gms

Sets
    i   planes      / 1*100 /
    r  runways      / 1*1 /;
    alias (i,j);
    alias (i,k);
Parameters
   E(i)  the earliest landing time for plane i
   T(i)  the target landing time for plan i
   L(i)  the latest landing time for plan i
   g(i)  the unit cost for landing before target time
   h(i)  the unit cost for landing after target time
   pi(i) the dual value for plane i
   node(i,j)  equals to 1 if j -> i or 2 otherwise
   S(i,j)    seperation time between i and j;
  Scalars
   pilast    the dual for the last equation ;
Variable RC;
Positive Variable x(i), a(i), b(i);
Binary Variable theta(i,j), col(i), y(i,j);
Equations
    Recost          define objective function
    ETime(i)            the earlest time for plane i
    LTime(i)            the latest time for plane i
    a1(i)               the relation between a and T x
    a2(i)               the relation between a and T E
    b1(i)               the relation between b and x T
    b2(i)               the relation between b and L T
    axb(i)              the relation between x and a b
    Comp1(i,j)          either i before j or j before i
    Comp2(i,j)          either i before j or j before i
    Comp3(i,j)          either i before j or j before i
    Comp4(i,j)          either i before j or j before i
    U(i,j)              seperation time requirment for all planes
    choose(i,j)         set node to be 1
    choose2(i,j)        set node to be 1
    delete(i,j)         set node to be 0
    delete2(i,j)        set node to be 0;

  Recost .. RC =e= sum(i, g(i)*a(i)+h(i)*b(i)-pi(i)*col(i)) - pilast;
```

```
  ETime(i) ..   col(i)*E(i) =l= x(i);
  LTime(i) ..   x(i) =l= col(i)*L(i);
  a1(i) ..      a(i) =g= col(i)*T(i)-x(i);
  a2(i) ..      a(i) =l= T(i)-E(i);
  b1(i) ..      b(i) =g= x(i)-col(i)*T(i);
  b2(i) ..      b(i) =l= L(i)-T(i);
  axb(i) ..     x(i) =e= col(i)*T(i)-a(i)+b(i);
  Comp1(i,j)$(S(i,j) < 9000) .. theta(i,j) + theta(j,i) =l= 1;
  Comp2(i,j)$(S(i,j) < 9000) .. theta(i,j) + theta(j,i) =g=
                                col(i)+col(j)-1;
  Comp3(i,j)$(S(i,j) < 9000) .. theta(i,j) =l= col(i);
  Comp4(i,j)$(S(i,j) < 9000) .. theta(i,j) =l= col(j);
  U(i,j)$(S(i,j) < 9000) ..     x(j) =g= x(i) + S(i,j)*theta(i,j)
       - (L(i)-E(j))*theta(j,i) - (col(i)*L(i)-col(j)*E(j))
       + (L(i)-E(j))*(theta(i,j)+theta(j,i));

  choose(i,j)$(node(i,j)=1) ..   col(i) =e= col(j);
  choose2(i,j)$(node(i,j)=1) ..  sum(k,theta(k,i)) =e=
                                sum(k,theta(k,j)) - col(i);
  delete(i,j)$(node(i,j)=2) ..   col(i)+col(j) =l= 1 + 1000000*y(i,j);
  delete2(i,j)$(node(i,j)=2) .   2 - 100000*theta(j,i) =l=
       sum(k,theta(k,j))-sum(k,theta(k,i)) +10000000*(1-y(i,j));

Model subproblem /all/ ;
$if exist matdata.gms $include matdata.gms
solve subproblem using MIP minimizing RC;
display x.l, a.l, b.l, col.l, theta.l, y.l;
$libinclude matout x.l I
$libinclude matout col.l I
$libinclude matout RC.l
```

Appendix D

Matlab programs for Branch-and-Price algorithm

```
function [Sequence,LandingTime,Cost,t] = Algorithm_air1_MinSub
% The entire branch-and-price algorithm
% The function air1_Branch() excute the column generation
clear all
tic
global  P E T L g h S oriE oriL
global A C Seq Time
global UB All_Nodes Nnode
p = 9;                                  % The number of problem
r = 2;                                  % The number of runways
% initialization ----- initialize the global parameters
initialization(p,r)
% definition of the output
Sequence = [];  LandingTime = [];   Cost = 0;
% The first parent node --- The LP relaxation of the problem
node_1 =[];     node_0 = [];
% Solve the root node by column generation
[Z,A,C,obj_val,Seq,Time] = air1_Branch(node_1,node_0);
% check if LP relaxation solution is integer
if floor(Z) == Z
    index = find(Z==1);
    for i = 1:length(index)
        Sequence = [Sequence; Seq(index(i),:)];
        LandingTime = [LandingTime; Time(index(i),:)];
        Cost = Cost + C(index(i),:);
    end
    return
end
% set the lower bound of the optimal solution
LP_LB = obj_val(end);
```

```
numberofcolumn = [length(C)]
% Branching and Bound
nodess = [];
delete = 0;
for n = 1:100
    % Update the information of the node to be explored
    [ro_1,co_1] = size(node_1);
    if delete==0 & ro_1 < (P-1)
    % choose a child node
        node = Node_choosing(Z,[node_1; node_0]);
        node_1 = [node_1; node];
        nodess = [nodess; node];
    else
    % remove the node and search for the next node
        num = find(nodess(:,1)==node_1(end,1)...
                   & nodess(:,2)==node_1(end,2));
        nodess = nodess(1:num,:);
        [row,col] = size(node_0);
        temp = [];
        for i = 1:row
            if ~isempty(find(nodess(1:num,1)==node_0(i,1) ...
               &nodess(1:num,2)==node_0(i,2)))
                temp = [temp; node_0(i,:)];
            end
        end
        node_0 = [temp; node_1(end,:)];
        node_1 = node_1(1:end-1,:);
    end
    delete = 0;
    % solve the linear relaxation problem for the current node by column
    % generation 'air1_Branch.m'
    [Z,A,C,obj_val,Seq,Time] = air1_Branch(node_1,node_0);
    numberofcolumn = [numberofcolumn length(C)]
    if obj_val(end) > UB+0.0001
        %% LP > UB, prune the node
        delete = 1;
    else
    %% LP <= UB , Check if it is integer or not
        if floor(Z) == Z
            %%% LP is integer
            if abs(obj_val(end)-LP_LB)<0.0001
                %% IP = LP_LB, then output the solution
                index = find(Z==1);
                for i = 1:length(index)
                    Sequence = [Sequence; Seq(index(i),:)];
                    LandingTime = [LandingTime; Time(index(i),:)];
                    Cost = Cost + C(index(i),:);
                end
                t = toc
                break
```

```
            else
                %% LP > LP_LB, update the UB, delete the node
                delete = 1;
                UB = obj_val(end);
                index = find(Z==1);
            end
        end
    end
end
% output the result
if isempty(index)
    for i = 1:length(index)
        Sequence = [Sequence; Seq(index(i),:)];
        LandingTime = [LandingTime; Time(index(i),:)];
        Cost = Cost + C(index(i),:);
    end
end
% the total running time
t = toc

function node = Node_choosing(Z,Branch)
% Choose the next node to be explored
global P R E T L g h S
global A C I Seq Time
Weight = zeros(P);
Z_I = find(Z~=0);
for i = 1:length(Z_I)
    for j = 1:length(find(Seq(Z_I(i),:)~=0))-1
        Weight(Seq(Z_I(i),j),Seq(Z_I(i),j+1)) = ...
        Weight(Seq(Z_I(i),j),Seq(Z_I(i),j+1)) + Z(Z_I(i));
    end
end
[ro,co] = size(Branch);
for m = 1:ro
    Weight(Branch(m,1),Branch(m,2)) = 0;
end
[I_1,J_1] = find(abs(Weight-1)<0.0001);
for i = 1:length(I_1)
    Weight(I_1(i),J_1(i)) = 0;
end
[II,JJ] = find(Weight==max(max(Weight)));
node = [[I_1,J_1]; [II(1),JJ(1)]];
```