Optimising airside operations of an airport for punctuality and stability

by

NINA HULLEBERG

THESIS

for the degree of

MASTER OF SCIENCE

(Master i Anvendt matematikk og mekanikk)

Faculty of Mathematics and Natural Sciences
University of Oslo

June 2014

Det matematisk- naturvitenskapelige fakultet
Universitetet i Oslo

Abstract

A key challenge in Air Traffic Management (ATM) is to provide a schedule with high throughput on the runways, and at the same time meet objectives connected to taxiing times and punctuality while ensuring safe operations within the apron, taxiway, runway, and terminal manoeuvring area. High throughput is achieved through optimised runway sequences. These sequences must frequently be revised, for example due to uncertainty in the available data. Hence, as updated information become available, the flight scheduling process continues throughout the day. Furthermore, since many of the activities and operations at the airport are prioritized and planned due to the previous schedule, it is important that the scheduling process does not create too much deviation from one plan to another.

In this thesis we present an approach for rescheduling, where we have modelled stability requirements in the objective function. We present new distance functions for measuring stability, and stability is formulated with respect to time and the runway sequence. We present computational results and analyse the trade-off between stability and optimality. Our experimental results indicate that including stability in the objective function greatly improves the stability without a major decrease in punctuality.

Acknowledgements

I would like to start by expressing my gratitude to my supervisors Dag Kjenstad, Carlo Mannino and Geir Dahl for inspiration, advice and guidance. I would also like to thank Marius Sandvik for all the discussions through the process, and MadMan for including me in their project at Sintef.

In addition, I would like to thank all my friends for not giving up on me, and a special thanks goes to Helene for valuable feedback, encouragement and smiles. Last, a warm thanks goes to my family for all love and support.

Nina Hulleberg
June 2014.

List of Abbreviations

Abbreviation	**Full Meaning**
AIBT	Actual In-Block Time
AMAN	Arrival Management
AOBT	Actual Off-Block Time
ATM	Air Traffic Management
ATOT	Actual Take-Off time
CTOT	Calculated Take-Off Time
DMAN	Departure Management
LP	Linear Programming
SESAR	Single European Sky ATM Reseach
SMAN	Surface Management
TIBT	Target In-Block Time
TOBT	Target Off-Block Time
TTA	Target Time of Arrival
TTOT	Target Take-Off Time

Contents

Chapter 1

Introduction

Air traffic management (ATM) is a process that involves strategic management of resources, with the goal of guiding aircraft safely and efficiently in the sky and on the ground. Over the past century there has been a huge increase in air traffic volume. In 2010, the European Air Traffic Management system controlled 9.5 million flights and recent forecast predict that this will increase to nearly 17 million flights per year by 2030 [13]. The possibility for airport expansions or new airports are limited, and as a consequence, the pressure on existing airports will be higher and the benefit of using optimisation technology will increase.

An important step towards successfully meeting the increased air traffic demand is to improve the efficiency of arrival and departure operations. In 2008, airport delays accounted for around 27% of the total delay in air transport network. The origin of airport delays is mainly related to the inefficiency of daily airport operations [12]. It is therefore important to provide a schedule with high throughput on the runways, and at the same time meet objectives connected to punctuality.

Typically, ATM considers three distinct problems: The Arrival Management Problem (AMAN), the Surface Management Problem (SMAN) and the Departure Management Problem (DMAN). These problems are tightly connected, with a common goal of creating an efficient and feasible airport flight schedule. Traditionally, the main objectives for an airport schedule have been punctuality and throughput. In addition there are many restrictions, for instance due to safety separation rules on the runway. Despite the complexity of the task and the short decision time available, most of the scheduling today is performed manually by the controllers.

However, the environment and the available information are rapidly changing, generating a need for revising the original schedule. Rescheduling may create a schedule that deviate significantly from the previous one. Since changes in the schedule need to be communicated from the controllers to the pilots, any change will increase the workload for the controllers. In addition, several planned activities are based on the original schedule and may be affected by the changes, and can cause confusion among the stakeholders. It may therefore be a good idea to keep the new solution close to the previous one. Loosely, this is what characterize *stability* in scheduling, and recently it has become a more common objective in ATM-scheduling.

Overview

The goal of this project is to find a mathematical model to represent the scheduling problem at an airport, and to develop an optimisation algorithm for solving it. Stability being a relatively new topic in scheduling, it lacks a common definition. In this thesis we will define a way to measure stability and analyse the trade-off between optimal solutions and solutions where we take stability into account. Hence, we set before us the following tasks:

- Present a small survey on scheduling and stability
- Define new distance functions for measuring stability
- Establish a linear programming(LP) formulation of a part of the scheduling problem
- Implement this formulation using an LP solver
- Perform simulations, analyse and document experimental results

The thesis is organized as follows:

Chapter 2 introduces some basic concepts and theory. This involves graphs, networks and linear programming. The theory from this chapter will be applied throughout this thesis. Chapter 3 will focus on scheduling and stability. There will first be a summary of issues that arise during a scheduling process, followed by a review of literature related to stability in scheduling, before stability distance functions are defined. My contribution to this chapter is the definitions of stability and stability performance measures.

In chapter 4 I present a mathematical description of the problem and a network scheduling model that will be used for numerical simulations.

In chapter 5 implementation details are discussed and an algorithm for solving the scheduling problem is developed. In Chapter 5, my contributions are the initial dual solution algorithm and the corresponding theorem and the discussion about how to reschedule. The approach presented here for finding an initial dual solution to a minimum cost flow problem have, as far as I know, not been presented in the literature before.

Chapter 6 presents some test runs using the algorithm from Chapter 5. All experiments and analysis are done by me.

Finally, I end this thesis with a summary and some concluding remarks in Chapter 7, and suggests some ideas for future work.

Collaboration with Sintef

This master thesis has been carried out in collaboration with the optimisation group at Sintef ICT, Oslo. The optimisation group is currently involved in building a decision support system that includes integrated arrival and departure management This work is done as part of SESAR (Single European Sky ATM Reseach), which is a European infrastructure modernisation programme. The program aims at developing a new generation of air traffic management systems.

Chapter 2

Background theory

Before we start exploring rescheduling and stability, we need some theory to build on. As will be seen later, our application build on the theory of flows in networks, and we will use a network-based implementation of the linear programming simplex method to solve the scheduling problem. In the first section, there will be an introduction to graph theory with central definitions. This section is based on theory from the book of Bertsimas and Tsitsiklis [3]. In section 2.2 we summarize some of the basic ideas and notation from linear programming and present the simplex method, before we look at flows in networks, and particularly the network simplex algorithm in section 2.3. The linear programming theory in these two sections are based on theory from Ahuja, Magnanti and Orlin [1], Vanderbei [14] and Bertsimas and Tsitsiklis [3].

2.1 Graph Theory

A graph $G = (V, E)$ is a mathematical structure used to represent relations between pairs of objects. It consists of two types of items: vertices (nodes) and edges (arcs). We let V denote the set of vertices and E denote the set of edges. The usual way to picture a graph is by drawing a circle for each vertex, and if there is an edge connecting two vertices, we draw a line between these two. An edge is either directed or undirected. In this thesis we will only work with directed graphs, i.e., the edges have a direction associated to them. We will illustrate directed edges using arrows, see figure 2.1 - 2.3.

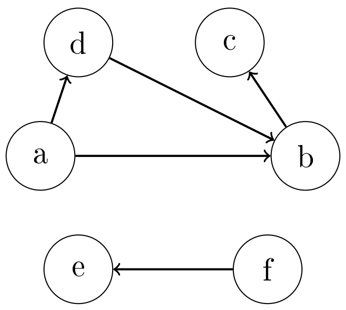


Figure 2.1: Disconnected Directed Graph $G = (V, E)$

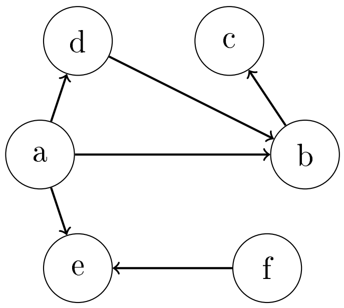


Figure 2.2: Connected Cyclic Graph $G = (V, E)$

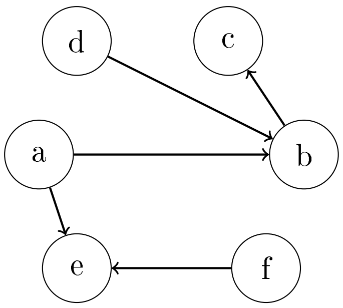


Figure 2.3: Connected Acyclic Graph $G = (V, E)$

We will denote vertex j with v_j, and if there is an edge $e \in E$ going from vertex i to vertex j, we will denote this edge by $e_{i,j}$ or simply (i, j). Here, i is called tale and j is called head. For the following definitions it is important to note that the edges can be traversed in either direction, irrespective of the given direction. For the edge (a, b) in figure 2.1, we say that (a, b) is a forward edge and (b, a) is a backward edge.

A walk is defined as a sequence $v_0, v_1, \ldots, v_k$ of vertices, that are linked together with an associated sequence of edges, $(0, 1), (1, 2), \ldots, (k-1, k)$. A walk is a path if all the vertices are distinct, and if $v_i = v_j$ for some i, j, we have a cycle. For instance, the vertices *a,b,d* in figure 2.1 and figure 2.2 create a cycle. A graph without a cycle is called acyclic. As in Bertsimas and Tsitsiklis [3], we allow a cycle to consist of only two distinct vertices. A walk, path or cycle is called directed if it contains only forward edges.

We say that a graph $G = (V, E)$ is connected if there is a path connecting every pair of vertices. For instance, both figure 2.2 and figure 2.3 are examples of graphs that are connected, while the graph in figure 2.1 is disconnected. If the graph $G = (V, E)$ is both connected and acyclic, then we say that the graph G is a tree, see figure 2.3.

Definition 2.1. (Spanning Tree) Given a directed graph $G = (V, E)$. A spanning tree $T = (V, E')$ is a connected subgraph of G with every vertex of the original graph, where T does not contain a cycle.

2.2 Linear Programming and Simplex Method

A linear programming (LP) problem is an optimisation problem, where both the objective function and the constrains are linear functions. In addition, there is often a set of non-negativity restrictions on the decision variables. A general linear programming problem with n decision variables x_j and m constraints can be written as

$$\begin{aligned} &\text{maximize} \sum_{j=1}^{n} c_j x_j \\ &\text{s.t} \qquad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad i = 1, \dots, m \\ &\qquad\qquad\qquad x_j \geq 0, \qquad j = 1, \dots, n. \end{aligned} \tag{2.1}$$

We should note that a linear program can be formulated in several ways. The problem can be written with both equality and inequality constraints, and the variables can be non-positive, non-negative or free (non-restricted). Very often it will be practical to represent the LP problem as a dictionary:

$$\begin{aligned} \xi &= \sum_{j=1}^{n} c_j x_j \\ w_i &= b_i - \sum_{j=1}^{n} a_{ij} x_j \qquad i = 1, \dots, m \end{aligned} \tag{2.2}$$

In a dictionary, the variables on the left side of the constraints are called basic variables, while those on the right side are called non-basic variables. The variables, w_i, are called slack variables or reduced costs, and represent the difference between the right-hand side and left-hand side in (2.1). The slack variables are by definition non-negative. Very often it is convenient to use the same notation for the slack and the decision variables. Therefore we often add them to the end of the list of x-variables:

$$(x_1, x_2, \dots, x_n, w_1, w_2, \dots, w_m) = (x_1, x_2, \dots, x_n, x_{n+1}, x_{n+2}, \dots, x_{n+m})$$

We say that a solution $\{x_j\}_{j=0}^{n}$ is feasible if all variables $x_0, x_1, \dots, x_n$ satisfies the constraints in (2.1). The set of all feasible points will constitute a convex region, which has a nice graphical representation. Each corner in the feasible region is called an extreme point. An extreme point is formed by the intersection of the lines corresponding to different constraints, and it cannot be a convex combination of two other feasible points. It can even be shown that every linear program always has an extreme point solution as one of its optimal solutions.

Definition 2.2. (Extreme point) Let P be a polyhedron. A vector $x \in P$ is an extreme point of P if we cannot find two vectors $y, z \in P$, both different from x, and a scalar $\lambda \in [0, 1]$, such that $x = \lambda y + (1 - \lambda) z$.

2.2.1 Duality theory

Linear programs come in primal/dual pairs. The two problems are closely connected, and each constraint in the primal has an associated variable in the dual. In the same way will each variable in the primal have an associated constraint in the dual. This gives us the dual problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{m} b_i y_i \\ \text{s.t} \quad & \sum_{i=1}^{m} y_i a_{ij} \geq c_j \qquad j = 1, \dots, n \\ & \qquad y_i \geq 0, \qquad i = 1, \dots, m \end{aligned} \tag{2.3}$$

It can be shown that a feasible solution for one of these two, will give a bound on the optimal objective function value for the other. This result is known as the Weak Duality Theorem (see for instance page 147 in [3] for proof):

Theorem 2.3 (Weak duality theorem). *If $(x_1, x_2, \dots, x_n)$ is feasible for the primal and $(y_1, y_2, \dots, y_m)$ is feasible for the dual, then*

$$\sum_j c_j x_j \leq \sum_i b_i y_i$$

The weak duality theorem has a number of consequences:

- If the primal problem has an unbounded solution, the dual problem is infeasible.
- If the dual problem has an unbounded solution, the primal problem is infeasible.
- Any feasible solution to a primal LP problem is a lower bound of the optimal value for the dual.
- Any feasible solution to a dual LP problem is an upper bound of the optimal value for the primal.

In fact, for linear programming there is never a gap between the primal optimal and the dual optimal objective value. This is usually referred to as the Strong Duality Theorem (see page 148-149 in [3] for proof):

Theorem 2.4 (Strong duality theorem). *If the primal problem has an optimal solution,*

$$x^* = (x_1^*, x_2^*, \dots, x_n^*),$$

then the dual also has an optimal solution,

$$y^* = (y_1^*, t_2^*, \dots, y_m^*),$$

such that

$$\sum_j c_j x_j^* = \sum_i b_i y_i^*.$$

Another important result from the duality theory, is the Complementary Slackness Theorem. This result gives us an important relation between the primal and the dual problem. For proof, se page 67 in [14].

Theorem 2.5 (Complementary Slackness Theorem). *Let $x = (x_1, x_2, \ldots, x_n)$ be primal feasible, and $y = (y_1, y_2, \ldots, y_m)$ be dual feasible. Let $(w_1, w_2, \ldots, w_m)$ denote the corresponding primal slack variables and $z_1, z_2, \ldots, z_n$ denote the dual slack variables. Then x and y are optimal for their respective problems if and only if*

$$x_j z_j = 0, \qquad \textit{for } j = 1, 2, \ldots, n$$

$$w_i y_i = 0, \qquad \textit{for } i = 1, 2, \ldots, m$$

2.2.2 Simplex method

The simplex method, developed by Dantzig in 1947, use the extreme point property to find an optimal solution. It is a well-known method and a widely used tool for solving LP problems. Here we will only give a brief outline and resent the basic ideas. For more details, we refer to appendix C in [1], chapter 3 in [3] or chapter 2–4 in [14].

Algorithm 1 Simplex method

```
Phase I:
    Step 1: Find a feasible point, $X$, or declare the problem infeasible.
Phase II:
    Step 2: Check for optimality. If current $X$ is optimal, STOP.
    Step 3: Pivot step. Return to step 2.
```

The simplex method is an iterative process, starting at a feasible point. After testing the optimality condition (is the solution both primal and dual feasible), it either stops or continues. If the solution does not fulfil the optimality criteria, the simplex method will perform an operation known as a pivot. A pivot means that a basic variable is replaced by a non-basic variable. Doing this, will give us a new solution with an objective value greater or equal to the initial one. When iterating we are moving from one dictionary to another. Since local optimality implies global optimality in linear programming, we do not need to check all possible solution, we will only inspect the extreme points. The simplex method will check all the adjacent extreme points and choose one of them as the next extreme point solution. This step is repeated until the current solution has the most desirable objective value compared to the adjacent points.

A final note to linear programs is that the simplex method will terminate with one of the following outcomes:

1. The problem has no feasible solution.
2. The problem has a feasible solution, but no optimal solution. The problem is unbounded, i.e. the optimal value is $-\infty$ (for minimization problems), or $+\infty$ (for maximization problems).
3. The problem is feasible and bounded, so the simplex method terminates with an optimal solution.

2.3 Network flow problems

The network flow problem is a special case of linear programming, and are among the most frequently solved linear programming problems. The problems are defined on graphs, so in this section we will use the theory from section 2.1.

2.3.1 Formulation of the network flow problem

A network is a directed graph where we have added some additional data. The added information is usually numerical data such as the external supply to each vertex $i \in V$ and cost per unit of flow along edge (i, j). Typically, there will be some vertices where flow can enter the network, and some where the flow can leave. For each $i \in V$, we let b_i denote the amount of material vertex i supply. We will use the convention that negative supply represent a demand. We say that vertex i is a source if $b_i > 0$, and vertex i is a sink if $b_i < 0$. If $b_i = 0$ we say that i is a transshipment vertex. $x_{i,j}$ will represent the flow from vertex i to vertex j. We will impose the following conditions on the flow along the edges:

$$\sum_{j:\,(k,j)\in E} x_{k,j} - \sum_{i:\,(i,k)\in E} x_{i,k} = b_k, \qquad k \in V \tag{2.4}$$

$$0 \le x_{i,j} \le u_{i,j}, \qquad (i,j) \in E \tag{2.5}$$

Equation (2.4) states that the amount of flow into a vertex must be equal to the total flow out of the same vertex. If we summarize both sides of this equation over all $i \in V$, we obtain $\sum\limits_i b_i = 0$. This means that the total demand equals the total supply in the network.

Equation (2.5) gives us the restrictions on a flow along an edge in the network. The flow must be non-negative and it cannot exceed the capacity $u_{i,j}$ of the edge. In this thesis we will assume that $u_{i,j} = \infty$ for all $(i, j) \in E$, meaning that the edges is uncapacitated.

2.3.2 Special cases of the network flow problem

The network theory provides a set of techniques for analysing graphs. In this section we will look at two special cases of the network flow problem, the shortest path problem and the minimum cost network flow problem.

Shortest path problem

Given a weighted graph $G = (V, E)$ with edge-cost $c_{i,j}$, we define the length of a directed path, p, as the sum of the costs of all edges on the path:

$$w(p) = \sum_{(i,j)\in p} c_{i,j}.$$

Then the length of the shortest path is given by:

$$l(u,v) = \begin{cases} \min\limits_{p \in P} w(p), & \text{where } P \text{ is the set of all paths from } u \text{ to } v \\ \infty, & \text{else} \end{cases}$$

This problem is very common in practice and it is often necessary to solve a shortest path problem before one can start to solve other and more advanced algorithms. There are several variations of the shortest path problem, but in this thesis we will only meet single-source shortest problem. The most important algorithms for solving this problems are Dijkstra's method (algorithm 2) and the Bellman-Ford algorithm (algorithm 3). Both algorithms are classified as label-setting algorithms. Of these two, Dijkstra is the most efficient one when it is implemented good, but it has a drawback since it does not work for graphs with negative weighted edges.

Algorithm 2 Dijkstra's Algorithm

```
Input: Directed graph $G = (V, E)$, Edgecost $c_{i,j} \geq 0$ and a root vertex r
for every vertex v
      $y_v := \infty$;
      $parent[v] := empty$;
end for
$y_r = 0$;
$Q :=$ the set of all vertices v in G
while $Q \neq \emptyset$ do
   $u := v \in Q$ s.t. $y_u$ is minimized
   $Q = Q \backslash u$
   if $y_u = \infty$ then
      $break$;
   end if
   for every neighbor v of u do
      if $y_v > y_u + c_{u,v}$ then
         $y_v = y_u + c_{u,v}$
         $parent[v] = u$
      end if
   end for
end while
return y
```

Minimum cost network flow

Another important problem in the study of networks is the minimum cost flow problem. Here we want to minimize the total cost of sending flow from the supply vertices to meet the demand at the sinks. The general minimum cost flow problem can be stated as:

Algorithm 3 Bellman-Ford Algorithm

```
Input: Directed graph $G = (V, E)$, Edgecost $c_{i,j}$ and a root vertex r
for every vertex v
    $y_v = \infty$;
end for
$y_r = 0$;
for $i =| V | -1$ do
  for every edge (u,v) in E do
    if $y_v > y_u + c_{u,v}$ then
      $y_v = y_u + c_{u,v}$
      $parent[v] = u$
    end if
  end for
end for
for every edge (u,v) in E do
  if $y_v > y_u + c_{u,v}$ then
    return Negative cycle
  end if
end for
```

$$\begin{aligned} \text{minimize} \quad & \sum_{(i,j)\in E} c_{i,j}x_{i,j} \\ \text{s.t} \quad & \sum_{j:(k,j)\in E} x_{k,j} - \sum_{i:(i,k)\in E} x_{i,k} = b_k, \qquad k \in V \\ & x_{i,j} \geq 0, \qquad (i,j) \in E \end{aligned} \tag{2.6}$$

We say that a solution $\overline{X} = \{x_{i,j} \mid (i,j) \in E\}$ is primal feasible if it satisfies all constraints in (2.6).

The minimum cost flow problem is one of the most fundamental of all network flow problems, and it has been studied extensively in the literature. There are several algorithms for solving this problem, see [1, 10] for detailed descriptions. In this thesis we will use a linear programming approach for solving the minimum cost flow problem. This approach and some important concepts connected to the network simplex algorithm will be presented in the next section.

As we noted in section 2.2, we can associate another related problem to every LP problem. For the minimum cost flow problem, the associated dual problem is:

$$\text{maximize} \quad \sum_{i\in V} b_i y_i$$

$$\text{s.t} \quad y_j - y_i + z_{ij} = c_{i,j}, \qquad (i,j) \in E \tag{2.7}$$

$$z_{i,j} \geq 0, \qquad (i,j) \in E \tag{2.8}$$

Note that (2.7) and (2.8) can be rewritten as

$$y_j - y_i \le c_{i,j}, \qquad (i,j) \in E \tag{2.9}$$

We say that a solution $\overline{Y} = \{y_i \,|\, i \in V\}$ is dual feasible if it satisfies all constraints in (2.9).

2.3.3 Network Simplex Algorithm

Since the minimum cost flow problem is a special class of linear programs, we could use the simplex method to solve it. However, the general simplex method does not take advantage of the underlying network structure. Therefore we will use a method that interpret the core concepts of the simplex method, and exploit the network structure. The method maintains a feasible spanning tree structure at each iteration and moves from one feasible solution to another feasible solution. The essential steps are given in Algorithm 4.

Definition 2.6 (Tree solution)**.** A flow vector $\overline{X}$ is called tree solution if it can be constructed by the following procedure:

1. Find a set $T \in E$ with $n-1$ edges that form a spanning tree
2. Let $x_{i,j} = 0$ for all $(i,j) \notin T$.
3. Use the flow balance equations to determine the flow variables $x_{i,j}$ for $(i,j) \in T$.

A tree solution is called a feasible tree solution if it also satisfies $\bar{X} \ge 0$.

In the section about linear programming (section 2.2), we talked about basic variables. In a network problem we will also refer to some variables as basic. We say that a variable $x_{i,j}$ is in the basis if the corresponding edge is in the tree solution.

Theorem 2.7. *A flow vector X is a basic solution if and only if it is a tree solution.*

For proof, see page 283 in [3].

Algorithm 4 Network Simplex

```
Input: G=(V,E), Edgecost $c_{i,j}$ and Vertex demand $b_i$
Require: $\sum_i b_i = 0$
Find an initial tree
Compute initial flow
Check for optimality ($x_{ij} \geq 0, z_{ij} \geq 0$).
if Optimal then
    Return $\sum_{(i,j)\in E} c_{ij}x_{ij}$
end if
while $z_{ij} < 0$ for some $(i,j) \in E$ do
    primal pivots due to the Primal Network Simplex Algorithm. Here we maintain and
    keep improving a primal feasible solution.
end while
while $x_{ij} < 0$ for some $(i,j) \in E$ do
    dual pivots due to the Dual Network Simplex Algorithm. Here the dual variables
    are updated to increase the value of the dual objective, while reduce the infeasibility
    of the complementary primal solution.
end while
```

Primal network simplex

The primal network simplex algorithm is used when the tree solution is primal feasible, but not dual feasible. The basic idea behind this method, is to pick an edge that is dual infeasible (i.e. $z_{i,j} < 0$) and let it enter the tree. Due to complementary slackness, $z_{i,j}$ will be increased to 0 and the edge (i, j) will become basic. When we let an edge enter a spanning tree, we will create a cycle, so to remain a tree solution, we have to remove another edge. The leaving edge is chosen from the set of edges in cycle that are oriented in the reverse direction as the entering edge. By repeating this procedure, we will move from one primal feasible solution to another primal feasible solution. The primal network simplex method can be summarized as:

Algorithm 5 Primal Network Simplex

```
while $z_{i,j} < 0$ for some edge $(i,j) \in E$ do
    • choose the edge with the smallest dual slack variable $\min z_{i,j}$, $(i,j) \in E$
    • Let the edge $(i,j)$ with the smallest dual slack enter the tree. With this edge
      added, there must be a cycle consisting of the entering edge and some of the
      other tree edges.
    • To remain a tree solution, we must remove one edge. The leaving edge is
      chosen from those edges on the cycle that go in the opposite direction from
      the entering edge. If there are more than one edge in the cycle pointing in the
      opposite direction, choose the one that have the smallest flow value.
end while
```

Dual network simplex

The dual network simplex algorithm is used when the tree solution is dual feasible, but not primal feasible. The basic idea behind this approach is to pick a tree edge that is primal infeasible and let it leave the spanning tree and become non-basic. By the definition of a tree solution, the flow on this edge will increase to 0, i.e. become feasible. This operation will split our solution into two subtrees. To maintain a spanning tree solution, we want to find an entering edge that will bridge the two subtrees into a spanning tree. Since we want the flow on the leaving edge to increase, we need to find an entering edge that bridge in the opposite direction from the leaving edge. The dual network simplex method can be summarized as:

Algorithm 6 Dual Network Simplex

```
while $x_{i,j} < 0$ for some edge $(i, j) \in E$ do
    • choose the edge $(i, j)$ with the most negative flow to leave the tree.
    • Let $(i, j)$ be the leaving edge. With this edge removed, our solution is now
      split into two subtrees.
    • To remain a tree solution, we must bridge these two subtrees. The entering
      edge must bridge in the opposite direction to the leaving edge. If there are more
      than one edge to choose between, choose the one with smallest dual slack.
end while
```

Chapter 3

Scheduling and stability

Scheduling is a decision-making process that is concerned with the allocation of one or multiple resources over a time period. There are a wide variety of situations in which schedules or plans are necessary, or at least useful, for example transportation schedules such as bus schedules. Scheduling theory can be described by the type of scheduling problems or by the methods used to find solutions. Today, there are a number of ways to attack these problems, and we usually divide the methods into two different categories, static and dynamic. Static, or off-line approaches assume that all information about the scheduling problem is known in advance, and does not change as the schedule is being computed or carried out. Dynamic or real-time approaches offer more flexibility as the scheduling take place as we go along, and we have the opportunity to revise and recompute the schedule. For both categories, the main objectives have been connected to makespan, tardiness, earliness and throughput, but recently new measures connected to the rescheduling process and the relationship between the new and the previous schedule have been developed.

In this chapter we will first summarize some questions that arise during a scheduling process, before we look more into one of the new measures, stability. In section 3.2 we will review some literature, particularly related to stability in scheduling and we end this chapter with section 3.2.2, where we define stability and generate stability distance functions that will be used in the mathematical model.

3.1 Scheduling

The airport environment is highly dynamic, and the available information are rapidly updated. In this section we will outline some of the questions that arise during a scheduling process.

How to schedule/reschedule?

The first question determines the way in which schedules are generated and updated. There are four related issues: scheduling scheme, scheduling horizon, type of response and performance metric.

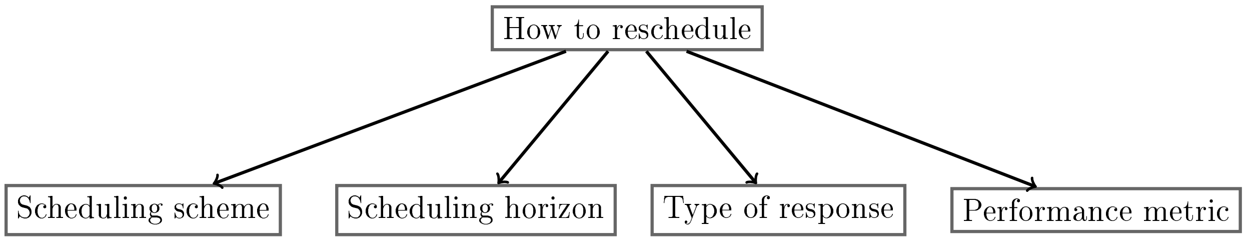


Figure 3.1: How to reschedule

The first issue is what approach we would like to use, i.e. do we want a static (off-line) or dynamic (on-line) approach. The second concern is the amount of data used during the schedule generation process. We often divide this into two groups: full scheduling, where all available information is used and partial scheduling, where only near-future information is used. The next issue is connected to how the process should react to changes. One approach is to do nothing, or one can perform a rescheduling. Last, we should decide on which performance metric to use. Traditionally, the classical performance measures (e.g. makespan and tardiness) have been preferred.

When to reschedule?

An initial schedule is the first schedule generated, often before the scheduling horizon begin. Unexpected disruptions and random events will disturb the system and generate a need to revise the schedule. The question about when to reschedule has to do with the timing and frequency of scheduling decisions, and the answer to this question will determine how fast the system will respond to disturbances.

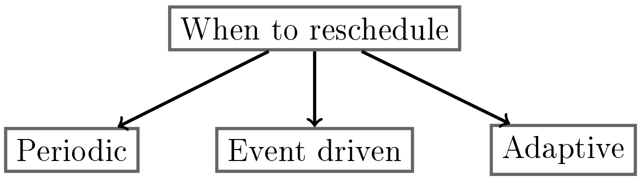


Figure 3.2: When to reschedule

There are several alternative ways to decide on the timing of scheduling decisions. The first alternative is to reschedule periodically, with either a constant or variable period length. Another approach is to revise the schedule after each major event change. A third way is to revise when a scheduling decision deviate more than a threshold form the original schedule.

Definition 3.1 (Realized schedule)**.** The schedule which is actually executed is called the realized schedule.

3.2 Stability

If you look up the word "stability" in a dictionary, you will find that it means that something is stable and resistant to change. Unfortunately, it does not exist a common definition of stability when it comes to scheduling, and traditionally, it has not been given a very high priority either. Stability is therefore considered as a relative new objective when it comes to scheduling, and is connected to the impact of disruptions induced by moving jobs and activities within the schedule. The distance or deviation between two schedules will be defined by a distance function. A schedule who does not deviate "much" from an earlier schedule, is called stable. As will be seen in the next subsection, when discussed in papers, stability has been concerned with the difference between the initial and the realized schedule. In this thesis we will mainly focus on the change in each rescheduling step, but we will also discuss the trend of change.

3.2.1 Stability in literature

Stability analysis of optimisation problems are motivated by the fact that the input data usually are given with some errors or not sufficient information. It is a relatively new performance measure, and has therefore not been given a major role earlier. However, the disturbance of moving jobs have been discussed under different names, e.g. "solution stability", "plan stability", "schedule stability" and "sensitivity". In recent years, the topic have started to attain attention from the research community, and there have been discussions about how to measure it and why. For instance, in Maria Fox et.al. [6] it is stated that preserving plan stability will reduce the cognitive load on humans working after the plan, and that plan stability will ensure coherence and constancy of behaviour, which in turn will lead to less stress.

One of the earlier studies in this area is by Wu et al. [15]. In this paper they consider a single-machine rescheduling problem with machine disruption. Their objectives are to minimize the schedule makespan, and at the same time achieve a schedule with high schedule stability. For stability, the authors consider two different performance criteria. The first is deviation with respect to job starting times and the second is a measure of sequence difference between two schedules.

In another study, Clark and Walker [4] consider nurse rescheduling with shift preferences and minimal disruptions. When it comes to stability, the authors consider two different approaches. The first approach is to minimize the number of changes, reasoning that fewer changes cause less disruption. In the second approach, each change is given a penalty and they want to minimize the sum of these penalties, reasoning that different types of change will cause different amount of disruption. Furthermore, there is a small discussion related to fairness among shift changes among the nurses. In addition, they suggest further research on the time of disturbance [4, page 161]:

> If there is frequent rescheduling then maybe penalties should be based on how far into the future a change is. For example, with a 4-week planning horizon, a change at the end of the 4th week may be less problematic than the same change occurring within the next few days. In fact, detailed scheduling of later weeks could be a waste of effort if this part of the schedule is never actually implemented.

A penalty approach is also presented in a paper by Petcu and Faltings [11]. They propose a general notation of solution stability, based on the cost of change from an already implemented solution to the new one. They argue that the number of changes between two solutions is irrelevant, and what matters is the total cost these changes induce. In this process they identify two kinds of commitments: soft commitments and hard commitments. Soft commitments can be revised and changed if the gain of changing is greater than the cost one has to pay for changing the current assignment to the new one. Hard commitments model irreversible processes and are impossible to undo (cost of changing is infinite). For uncommitted variables, the cost of changing the current assignment is 0.

When it comes to ATM-optimisation, James Adam David Atkin is one of the pioneers. In his PhD thesis from 2008 he argues for sequence stability [2, page 105]:

> There are often cases where there are multiple sequences with very similar costs. In this case, it is better to favour a previously used sequence rather than allow sequence changes that will have little benefit. This is especially important if there is some uncertainty in the data used to make the decisions as small perceived benefits may be purely down to data errors.

He also argues for penalty costs [2, page 170]:

> Not all change has the same cost. For example, the early part of the sequence is important as these are usually the aircraft that are already within the holding area and under the control of the runway controller. These are the only aircraft which would have already been given instructions. It is important that at least the early part of the sequence is stable over time, rather than constantly fluctuating between wildly different sequences of similar cost.

In practical all papers, stability is considered as a characteristic that reflects the degree the schedule information changes over time, but it does not exist a common definition. It usually refers to the relationship between two solutions and the discussion is often limited to the phase of the algorithm, where an initial solution has already been found and additional calculations are being performed. To end this section, here are some of the performance measures used in literature to define the distance function to measure the distance between two solutions:

- The number of variables with changed value
- The percent of recalculations required
- The percent of variables that has changed value
- The number of perturbations minus the number of input permutations

- The total difference between the completion (or starting) times in the two solutions
- The average difference between the completion (or starting) times in the two solutions

3.2.2 Definition of stability

In this thesis we will address the problem of rescheduling after a disruption has occurred. A disruption could be that a flight is delayed or even cancelled, a runway could be closed or a new flight need to be added to the schedule. Due to safety reasons, any of these events will provoke the need of revising the schedule. We will therefore use a reactive approach, where the rescheduling process is event driven to maintain a feasible and safe schedule.

While the flights are still at the stand, there is a great deal of flexibility in the sequence. However, as fast as the aircraft has left the gate, the resequencing is limited to the holding areas and usage of multiple runway entrances. Therefore, the early part of the sequence is especially important since this part of the schedule contains aircraft that have already started and have been given instructions. Our scheduling model, which will be presented in chapter 4, will capture sequence changes that are not possible due to the airport structure and resource limitations. This will be illustrated by an example in section 5.5. Nonetheless, any resequencing of aircraft will increase the workload for the controllers. For this reason, we would like the stability objective to capture the performance of the controllers and what kind of changes they are capable of communicating in short time.

For stability we will use two distinct measures, one connected to time and another connected to sequence. We will measure the schedule change, using the previously calculated schedule as a baseline.

Time stability

The discussion above brings us back to our problem, and it is time to define our first stability measure. We say that a schedule, s_i, is absolute time stable if the planned activities in schedule s_i are scheduled to the same time as in the previous plan, s_i. This gives us a way to measure time stability :

Definition 3.2 (Time stability distance function)**.** Assume we have a series of schedules, $s_0, s_1 \dots s_i$, where s_0 is the initial schedule. Let t_j^f be the time flight f enter airport resource j in schedule s_i, and let τ_j^f be time flight f enter resource j schedule s_{i-1}. Let $\zeta_j^f = \max(0, t_j^f - \tau_j^f)$ and $\varepsilon_j^f = \min(0, t_j^f - \tau_j^f)$. The distance function for time stability is given by

$$\sum_{f \in F} \sum_{j \in P_S^f} b_j^{\varepsilon f} \varepsilon_j^f + b_j^{\zeta f} \zeta_j^f \,,$$

where $b_j^{\varepsilon f}$ and $b_j^{\zeta f}$ is the costs of deviate from the previous value and P_S^f the set of resources where we want to measure stability for flight f.

Definition 3.3 (Time stable)**.** Assume we have a series of schedules, $s_0, s_1 \dots s_i$, where s_0 is the initial schedule. Given the time stability distance function, we say that s_i is time stable if the time deviation cost between the planned flight activities in plan s_i and s_{i-1} are within a given threshold, T. That is

$$\sum_{f \in F} \sum_{j \in P_S^f} b_j^{\varepsilon f} \varepsilon_j^f + b_j^{\zeta f} \zeta_j^f \leq T$$

Sequence stability

Only measuring time deviation may reflect the changes in the sequence, but a time change does not necessarily mean that the sequence has changed. Since some of the communication between the controllers and the pilots involve information about other aircraft, the workload for the controllers may also increase if there are sequence changes.

When measuring sequence changes, there are two main approaches:

- Absolute sequence position
- Relative sequence position

In the first approach, the importance is connected to the actual position in the sequence (e.g.. flight f is third in line), while the second approach emphasize the relative position (e.g. flight g is behind flight f). In communication between the controllers and the pilots, information about the sequence is conditional and we will therefore use a relative sequence positioning approach.

When measuring sequence stability on the runway, we will look at two following schedules, s_i, s_{i-1} and count the number of flights that has a different leading flight in the two schedules. For this we will use the Hamming distance H[1].

Definition 3.4 (Sequence stability distance function)**.** Assume we have a series of schedules, $s_0, s_1 \dots s_i$, where s_0 is the initial schedule. For each schedule, there is given a sequence vector X, where

$$x_{i,j} = \begin{cases} 1, & \text{flight } i \text{ is the leading flight of } j \\ 0, & \text{else} \end{cases}$$

Then the sequence distance between schedule s_{i-1} and s_i are given by $\frac{1}{2}H(X^{i-1}, X^i)$

Definition 3.5 (Sequence stable)**.** Assume we have a series of schedules, $s_0, s_1 \dots s_i$, where s_0 is the initial schedule. Given the sequence stability distance function, we say that s_i is sequence stable if $H(X^{i-1}, X^i) \leq T$, where T is a given threshold.

[1]The Hamming distance of two vectors of equal dimension is equal to the number of coordinates in which they differ

Chapter 4

The model

In the introduction we described the goal of the project, and using the theory from chapter 2 and 3, we are now ready to go a little further. We will first look at some airport terminology, before we investigate the formulation of the scheduling problem in section 4.2. In the two following sections we consider the objectives and the constraints. In section 4.5 we summarize the variables and present the entire problem as one linear program. In this section we will also associate a special graph to the problem that will be used when solving the problem.

4.1 Airport terminology

In this section we will formalize some airport terminology that will be used in this thesis.

The flight schedule is the driving force at an airport. In the schedule there is information about all flight movements during a time horizon H. The air traffic controllers are responsible for the movements of airplanes at the airport. The responsibility is divided between the Clearance controller, which provide flight plan, an Apron controller, which give instructions for pushback, a Ground controller, responsible for the taxiing, and a Runway controller, responsible for the runway.

4.1.1 Airport

An airport is divided into several areas. The airside include all areas used by aircraft. A gate is where passengers board and disembark a plane, while a stand is more general and refer to an area where aircraft are parked. A runway is a designated area used for take-offs and landings, while taxiways are roads that connect the parking areas and the runways. A manoeuvring area includes both runways and taxiways.

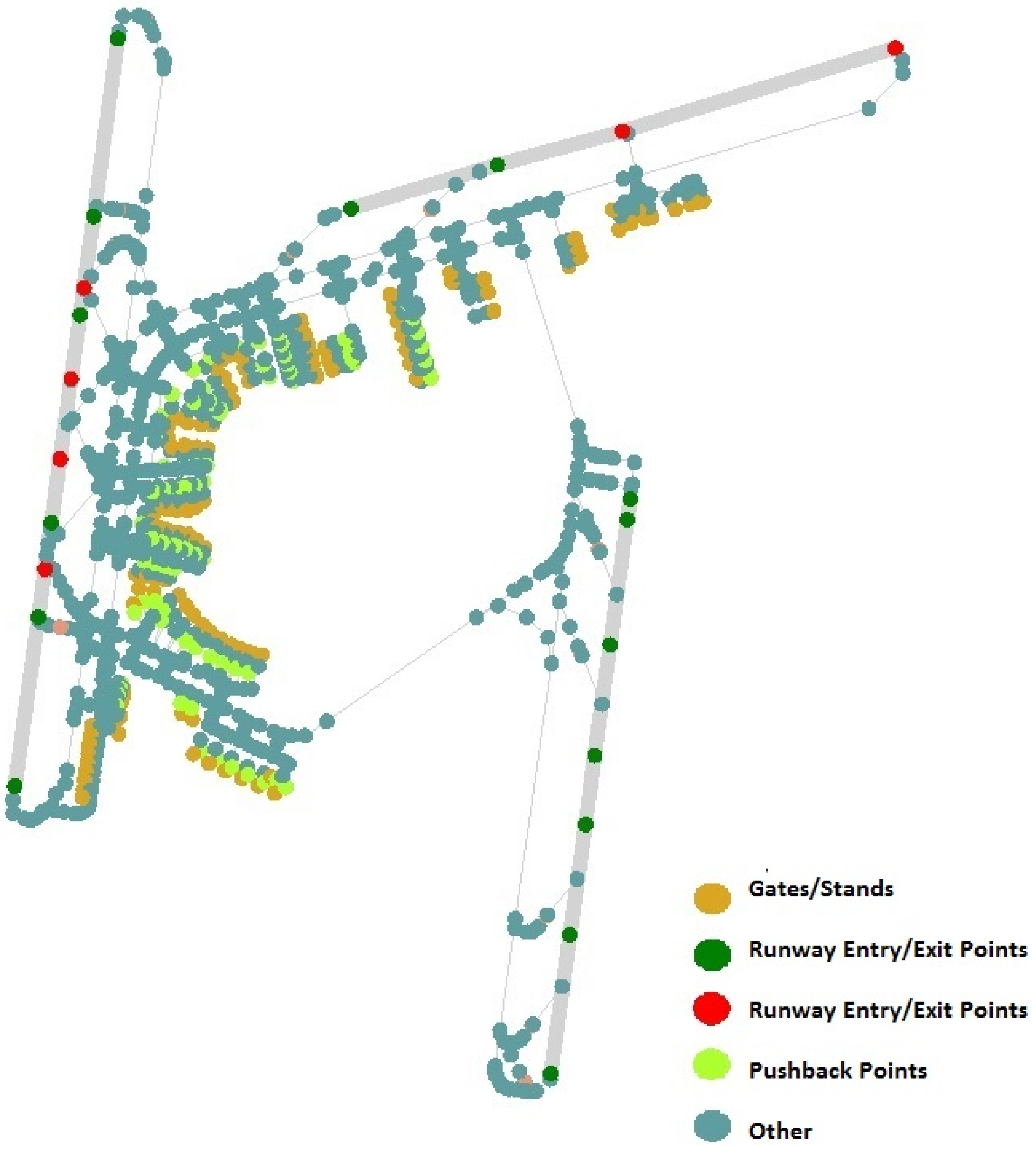


Figure 4.1: Airport chart - Arlanda

This thesis examines situations at the airport Stockholm-Arlanda, where a number of flights are supposed to arrive and departure. The airport is represented by a directed graph $G = (V, E)$, where the vertices represent places or airport resources (such as stands and runway exits), and the edges represent airport segments. The airport resources at Arlanda are represented by 1025 vertices, and the airport segments by 2522 edges. To distinguish between the different graphs used in this thesis, we will refer to this graph as an airport graph.

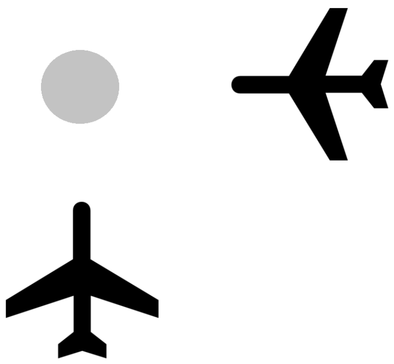


Figure 4.2: Resource conflict

We will divide the resources into two categories, holding points and non-stop points. A holding point is a resource where aircraft are permitted to stop, while they must drive straight through a non-stop point. In figure 4.2, two flights are approaching the same resource. Since two distinct flights cannot occupy this resource simultaneously, we say that this resource is non-shareable. A schedule where no flights occupy the same resource simultaneously is called conflict-free. For all non-shareable resources, a decision on "who goes first" must be taken. Together, these decisions will give us a sequence of the order in which the flights occupy the resource. We will call this sequence a precedence sequence.

4.1.2 Flights

Associated with each flight f, there is a set of information used by the air traffic controllers to handle the flow of traffic. This information includes airline, call sign of flight and size, but it also contains information about gate allocation and major events for the flight. A major event is often called a milestone and is a significant event that occur during the planning or operation of a flight. Each milestone has an associated target time, which represent the time we want the milestone to be achieved.

A pushback is when a flight f backs up from the gate, and take-off is the phase when a flight goes from the ground to flying in the air. When a flight departs from its parking position, we say that the flight is off-block. Flight f is called a departure or a departing flight if it starts at a gate, does a pushback and then taxi to a runway and perform a take-off. A departing flight has mainly two milestones connected to its route: off-block and take-off. The target off-block time (TOBT) is the time when f should be ready to leave the gate. In the same way, f is given a target take-off time(TTOT), which is the time f is desired to take off. The actual off-block time(AOBT) and the actual take-off time(ATOT) represent the time f actually leaves the gate and the time it takes off. The take-off time is measured at a take-off point.

If f is an arriving flight, f is approaching the airport in the air, perform a landing and continues to a gate. The estimated time of arrival, is a prediction of when flight f will land, and the actual landing time is when f actually touch the ground. This point is called a touch-down point. Landing is the most common milestone for arriving flights, with the possibility of also measuring the in-block time. In-block time is the time the flight arrives at its parking position. In that case, the actual in-block time (AIBT) is the actual time flight f puts the parking brakes on at a gate.

Airport slots are specific time periods when an aircraft is permitted to land or take off. The most common time slot is a fifteen minute window, where five minutes are before the target time and ten after. Airport slots are often called target windows.

In this thesis we will only work with one runway, used for both arrivals and departures, and we will only consider flights taxiing between the runway and the gate, meaning that for instance flights on maintenance will not be considered.

4.2 The problem

Given the airport graph of Arlanda, $G = (V, E)$, and information about a set of flights that will arrive and depart during a time horizon H, we want to find an initial conflict-free schedule that minimize the deviation from the given target times. Further, we will revise the schedule when the available information is changed and recompute the solution with the goal of finding a feasible schedule that maintains stability and minimize the deviation from the given, and possibly updated, target times.

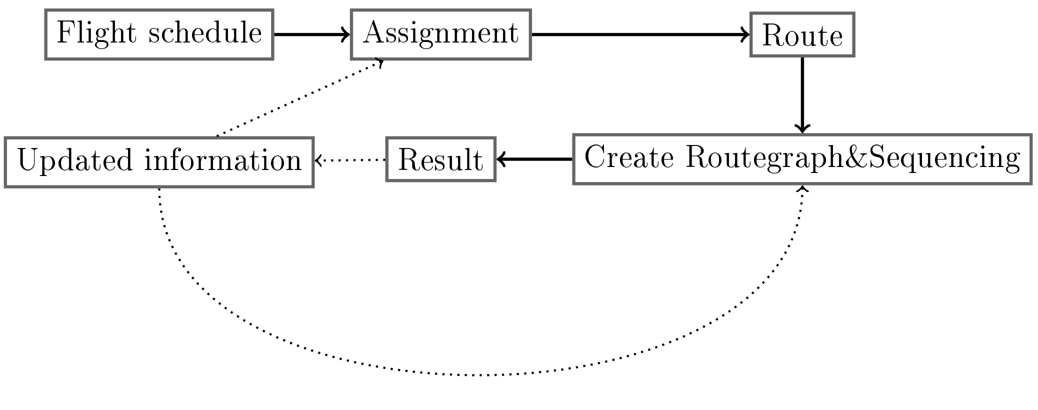


Figure 4.3: The Scheduling Process

We let $F = L \cup D$ be the set of flights that will arrive and depart during a time horizon H, where L is the set of arrival flights and D is the set of departure flights. For each flight $f \in F$, the arrival or departure gate is assigned, and there is a set of feasible routes, $R(f)$, from the initial position of f to its destination. A flight route, r^f, is simply a sequence of vertices, $r^f = (v_0, v_1, \ldots, v_k)$, where $v_0, v_1, \ldots, v_k \in V$. The goal is to compute a schedule vector $t^f = (t^f_{v_0}, t^f_{v_1}, \ldots, t^f_{v_k})$ for each flight f, where t^f_i is the time flight f start using resource i. Meaning that t^f will contain information about the time which a flight $f \in F$

should enter every point of its route r^f. These schedule vectors are what we refer to as a schedule.

For every edge $(u, v) \in E$, and every flight $f \in F$ we let $c^f_{u,v}$ be the running time for f through edge (u, v). We assume that the flights will traverse the edges in fixed time, and cannot stop within an edge, meaning that waiting is only allowed at vertices. For non-shareable resources, a precedence decision need to be taken.

The objective function will include several terms. The first objective is to minimize the deviation from the given target times. The second objective will only be used when we revise the schedule. This objective is to minimize the deviation from the previous schedule. Deviation from the previous schedule or from target times will be given a cost. We will call the total cost given by the objective function for schedule cost.

The problem can be summarized as follows:
Problem: For all flights $f \in F$, find the shortest route in $R(f)$, generate a precedence sequence at all non-shareable resources, find a feasible schedule such that the schedule cost is minimized.

In order to tackle this problem, we decompose it into four solution steps:

1. Find for all $f \in F$ the shortest legal route $r^f \in R(f)$
2. For all non-shareable resources, find a precedence sequence and associated precedence constraints
3. Compute a conflict-free solution, respecting the context established in step 1 and 2
4. If new information appear, revise the schedule

In the next section we will find an LP model to represent the problem in step 3, given the context established in Step 1 and 2. Step 1, 2 and 4 are discussed in Chapter 5.

4.2.1 Presentation of a simple airport example

Throughout this chapter, we will use a small and simple airport to illustrate algorithms and key points. This airport has one runway and two stands, and the runway can only be traversed from left to right. In total the airport consist of 7 resource-points (see figure 4.4):

1. = Runway entry.

   Holding point only for departing flights.

2. = Runway exit.

   Holding point only for arriving flights.

3. = Holding area.

   Holding point for arriving and departing flight

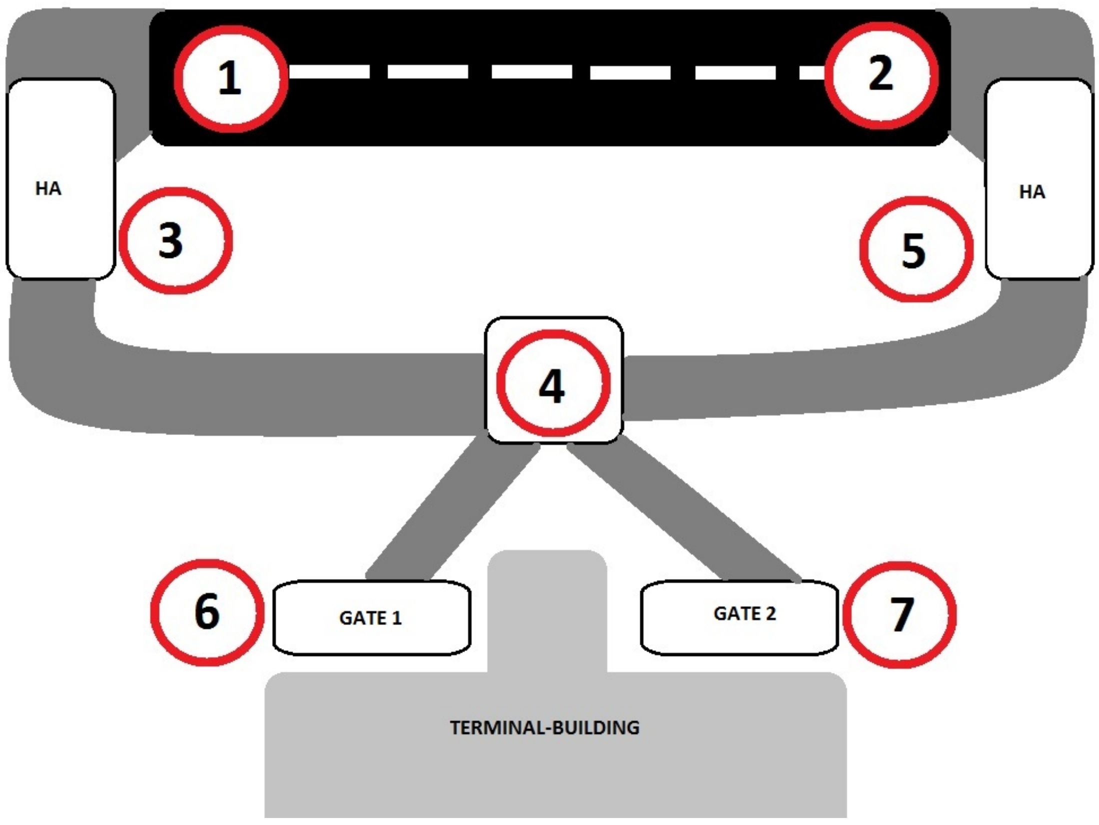


Figure 4.4: Airport chart - Fictive Airport Example

4. = Crossing point.

   Non-stop for arriving and departing flights

5. = Holding area.

   Holding point for arriving and departing flights

6. = Gate 1.

   Holding point for arriving and departing flights

7. = Gate 2.

   Holding point for arriving and departing flights

We will assume that the running time between two airport resources are fixed and equal for all flights f:

$$c^f_{1,2} = 60 \qquad c^f_{5,4} = 25$$
$$c^f_{2,5} = 10 \qquad c^f_{4,6} = 40$$
$$c^f_{3,1} = 10 \qquad c^f_{7,4} = 40$$
$$c^f_{4,3} = 35$$

4.3 Constraints

To be executable, the final schedule must satisfy taxi route, precedence and separation constraints. A schedule satisfying all given constraints are called feasible.

4.3.1 Taxi route constraints

As outlined in the last section, each flight $f \in F$ is assigned a taxi route between the runway and the gate, and we will see how in section 5.1. Let P_R^f be the set of airport resources flight f visits on its route. The route takes turning restrictions, one-way streets and other restrictions into account.

Along this taxi route for a flight f, there are some holding points (P_H^f) and some points flight f must drive straight trough $(P_{\not{H}}^f)$. This gives us two types of constraints, that every feasible schedule must satisfy, to represent the relation between the entering time at two following distinct resources:

$$t_j^f \geq t_i^f + c_{i,j}^f, \qquad i \text{ is a holdning-point} \tag{4.1}$$

$$t_j^f = t_i^f + c_{i,j}^f, \qquad i \text{ is a non-stop point} \tag{4.2}$$

These equations and inequalities will make sure that the airport resources are visited in order, and that the arriving time at resource j cannot be earlier than the arriving time at resource i plus the time it takes to drive between them.

Example 4.1. ***Small (fictive) airport example with two flights:***
Assume that one flight will land and another will take off at the airport illustrated in figure 4.4. The scheduling horizon will start at 0. Flight A is the arriving flight and A will land on the runway after 65 time units. Then it will follow the route $1-2-5-4-6$. *Flight B is a departure and starts taxing from gate 2. Flight B will be ready to start taxing after 70 time units and it will follow the route* $7-4-3-1-2$.

This information can be described by the following constraints:

$$\begin{array}{c} \underline{\textit{Flight A:}} \\ t_1^A + 60 = t_2^A \\ t_2^A + 10 \leq t_5^A \\ t_5^A + 25 \leq t_4^A \\ t_4^A + 40 = t_6^A \end{array} \qquad\qquad \begin{array}{c} \underline{\textit{Flight B:}} \\ t_7^B + 40 \leq t_4^B \\ t_4^B + 35 = t_3^B \\ t_3^B + 10 \leq t_1^B \\ t_1^B + 60 \leq t_2^B \end{array}$$

Upper and lower bounds

In the example above, there are also given information about when each flight enter the system. Observe therefore that some schedule variables need to satisfy a lower or upper bound, $L_i^f \leq t_i^f \leq U_i^f$, where L_i^f is the earliest time flight f can arrive at resource i, and U_i^f the latest.

Example 4.1 (continued) *There are two variables that are assigned bounds, t_1^A and t_7^B. Flight A will land at vertex 1 65 time units after the time horizon H has begun, giving us the following constraint:*

$$t_1^A = 65.$$

Associated with flight B, we have an earliest off-block time. This time is given as

$$t_7^B \geq 70$$[I]

4.3.2 Separation Rules

At Arlanda, there are several separation rules that need to be followed, both on the taxiways and on the runway. In general, these rules are guidelines for safe manoeuvring at the airport.

Precedence constraint

When two flights, $f, g \in F$ need to access the same resource i, a decision on "who goes first" must be taken. If f uses resource i before g, then $t_i^f \leq t_i^g$. Likewise if g goes first, then $t_i^g \leq t_i^f$. Since two flights can not occupy the same resource at the same time, the first flight has to leave i before the second flight can enter. Therefore we need a way to represent the time a flight leaves a resource. Let $z_{i,j}^f$ be the holding time for flight f at resource i, before it continues against the next resource point, j. Then $t_i^f + z_{i,j}^f$ will represent the time flight f leaves resource i. Notice that $t_i^f + z_{i,j}^f = t_j^f - c_{i,j}^f$.

In addition we add a proper time separation at i, $s_i^{f,g}$, between the two flights. We therefore have the following precedence constraints:

$$t_i^f + s_i^{f,g} \leq t_i^g, \qquad \text{when } i \text{ non-stop point} \tag{4.3}$$

$$t_j^f - c_{i,j}^f + s_i^{f,g} \leq t_i^g, \qquad \text{when } i \text{ is a holding point for f} \tag{4.4}$$

Vortex and radar separation

In addition to the precedence constraints that applies to every airport resource, there are some separation rules that are particularly connected to the sequence on the runway.

[I]Remember that t_i^f represents the time flight f enters resource i. For departing flights we are often interested in the off-block time (the time an aircraft departs from its parking position). The off-block time can be calculated from the entering time at the next position in the taxiroute. How to compute the time flight f leaves resource i is described under Precedence constraints in section 4.3.2

The total separation between two flights are decided by a required separation due to turbulence and a required radar separation.

Different types of aircraft, depending on their weights, generate different amounts of turbulences, also known as wake turbulence or wake vortices. The turbulence is especially hazardous in the region behind an aircraft in the take-off or landing phases of flight, and can cause problems for the following aircraft. We therefore require a minimum separation time between flights.

The vortex separation time is based on the maximum take-off mass, and every aircraft is placed into a category. The general rule is that an aircraft of a lower wake vortex category must not be allowed to take off less than two minutes behind an aircraft of higher wake vortex category. For arriving flights, the required radar separation is often greater than the required vortex separation time. The radar separation requires separation between flights both horizontal and vertically.

At Arlanda the separation rules in table 4.1 are used, but most of the aircraft in use at this airport, are categorized in the same size. We have therefore simplified the rules as given in table 4.2.

D-D:	*a*) If the same departure procedure is used and the speed of the the trailing flight is higher	150 sec
	b) If a heavier aircraft is in front of a lighter aircraft	120 sec
	c) Otherwise	60 - 80 sec
D-A:	From the start of the departure roll until the arriving flight starts landing	78 sec
A-D:	The start of departure roll can start after the runway is free	30-50 sec
A-A:	*a*) If the first flight is heavier than the second	128 sec
	b) Otherwise	78 sec

Table 4.1: Vortex Separation table

First Aircraft	Second Aircraft	Runway Separation
A	A	80 seconds
A	D	40 seconds
D	A	80 seconds
D	D	80 seconds

Table 4.2: Generalized Separation Rules

With these rules in mind, the separation on the runway are particularly interesting. There will be precedence constraints for entering the runway, and there will be safety separations for take-off and landing. Depending on the sequence, the separation constraints may vary a little. Notice that the runway entry is a holding point for departing flights, while the runway exit is a holding point for arriving flights. Based on the precedence constraints (4.1, 4.2), we get the runway separation constraints in table 4.3.

The separation constraints in table 4.3 are illustrated later in this chapter, see figure 4.8 - 4.11.

First Aircraft	Second Aircraft	Separation constraint
Arrival	Arrival	$t_E^2 \geq t_T^1 - c_{E,T}^1 + s_E^{1,2}$
Arrival	Departure	$t_E^2 \geq t_T^1 - c_{E,T}^1 + s_E^{1,2}$
		$t_{IN}^2 \geq t_{IN}^1 + s_{IN}^{1,2}$
Departure	Arrival	$t_E^2 \geq t_T^1 + s_E^{1,2}$
Departure	Departure	$t_E^2 \geq t_T^1 + s_E^{1,2}$
		$t_{IN}^2 \geq t_{IN+1}^1 - c_{IN,IN+1}^1 + s_{IN}^{1,2}$

Table 4.3: Separation constraints on the runway

Example 4.1 (continued) *In addition to the taxi route constraints we have special separation constraints. For simplicity, here are two examples that will illustrate both precedence and safety constraints discussed above:*

- *Flight B has to pass vertex 4 at least 20 time units before flight A.*
- *Flight A has to exit the runway at least 45 time units before flight B can start driving on the runway (i.e. leave vertex 1)*

Giving us the following constraints:

$$\begin{aligned} t_4^B + 20 &\leq t_4^A \\ t_2^B - 60 &\geq t_2^A + 45 \end{aligned}$$

which can be rewritten as

$$\begin{aligned} t_4^B - t_4^A &\leq -20 \\ t_2^A - t_2^B &\leq -105 \end{aligned}$$

4.4 Objectives

Our objective function will include several terms, divided into two categories: punctuality and stability.

4.4.1 Punctuality

The first goal is to minimize deviation from wanted target times. In aviation the most important target times are: target off-block time, target take-off time and target time of arrival (TTA). We define off-block, take-off and landing as milestones and the milestone position is where we measure the actual time for these events. Usually, the actual off-block time will be measured at the gate, the actual take-off time at a take-off point and the actual landing time at a touch-down point. Since the take-off point and the touch-down point may vary from flight to flight, we have simplified this and the runway exit will be the point where we measure the actual take-off time and the actual landing time will be measured on the runway entry. We define P_M^f be the set of milestone positions for flight f, and we let T_i^f denote the target time for flight f at milestone i. Then the deviation is given by:

$$|t_i^f - T_i^f|, \qquad i \in P_M^f, f \in F$$

Observe that $t_i^f - T_i^f$ will be negative when flight f arrives early at i and positive when f is delayed. There can be reasons to give different penalty cost for deviation, depending if the flight is early or late. We therefore define the following variables:

$$\delta_i^f = \max(0, t_i^f - T_i^f) \tag{4.5}$$

$$\epsilon_i^f = \min(0, t_i^f - T_i^f) \tag{4.6}$$

where δ_i^f represent tardiness and ϵ_i^f earliness for flight f at a milestone i. For each time unit flight f deviates from target time at i, there is an associated cost. If the flight arrives early the cost per time unit is given by $b_i^{\epsilon f}$, and if the flight arrives late the cost per time unit is given by $b_i^{\delta f}$. By definition $\delta_i^f \geq 0$ and $\epsilon_i^f \leq 0$, which implies that $b_i^{\delta f} \geq 0$ and $b_i^{\epsilon f} \leq 0$. Then, the objective function is given by:

$$\min \sum_{f \in F} \sum_{i \in P_M^f} \left(\delta_i^f \, b_i^{\delta f} + \epsilon_i^f \, b_i^{\epsilon f} \right)$$

.

The next step is to translate the non-linear expressions for earliness and tardiness into a set of linear equations:

$$\delta_i^f \geq t_i^f - T_i^f \tag{4.7}$$

$$\epsilon_i^f \leq t_i^f - T_i^f \tag{4.8}$$

$$\delta_i^f \geq 0 \tag{4.9}$$

$$\epsilon_i^f \leq 0 \tag{4.10}$$

Equations (4.9) and (4.10) make sure that at least one of δ_i^f and ϵ_i^f will be zero, since (4.5) and (4.6) can not be non-zero simultaneously.

Example 4.1 (continued) *In our example, the final destination for flight A is vertex 6, while the final destination for flight B is vertex 2. Each flight f has a target time T_i^f for arriving at resource i. The deviation for flight A is given by $(t_6^A - T_6^A)$, and for flight B the deviation is given by $(t_2^B - T_2^B)$. With the above definition, the deviation constraints for our airport example will be:*

Flight A:	*Flight B:*
$\epsilon_6^A \leq t_6^A - T_6^A$	$\epsilon_2^B \leq t_2^B - T_2^B$
$\delta_6^A \geq t_6^A - T_6^A$	$\delta_2^B \geq t_2^B - T_2^B$
$\epsilon_6^A \leq 0,\ \delta_6^A \geq 0$	$\epsilon_2^B \leq 0,\ \delta_2^B \geq 0$

In the example above, we are only considering whether there is a deviation from target-time, and since we are measuring both the tardiness and the earliness, we have the opportunity to set different cost whether the flight is early or late. At real airports, departing flights are often given a take-off window or time-slot, in which the flight is permitted to take-off. The window usually starts five minutes before the target take-off time and ending ten minutes after. If a flight misses its slot, the flight is considered dropped and can be given a new slot later that day. Dropping a flight will give additional cost.

In this thesis we will model this by adding an extra cost if a flight f reaches its milestone outside the target window. The target window is given by $[T_i^f - l_i^f, T_i^f + u_i^f]$, where l_i^f and u_i^f represent the distance between the bounds of the target window and the target time. If f arrives i later than $T_i^f + u_i^f$, this will be represented by δ_{ui}^f, and if f arrives earlier than $T_i^f - l_i^f$, the extra earliness is represented by ϵ_{li}^f. This gives us the following equations:

$$\begin{aligned} \delta_{ui}^f &= \max(0, \delta_i^f - u_i^f) \\ \epsilon_{li}^f &= \min(0, l_i^f + \epsilon_i^f) \end{aligned} \tag{4.11}$$

In the same way as the deviation equations, these can be transformed into a set of linear equations:

$$\begin{aligned} \delta_{ui}^f &\geq (\delta_i^f - u_i^f) \\ \epsilon_{li}^f &\leq (l_i^f + \epsilon_i^f) \\ \delta_{ui}^f &\geq 0 \\ \epsilon_{li}^f &\leq 0 \end{aligned} \tag{4.12}$$

4.4.2 Stability

The second objective, stability, was introduced in chapter 3. We will use two distinct measures, one connected to time and another connected to the flight sequence on the runway.

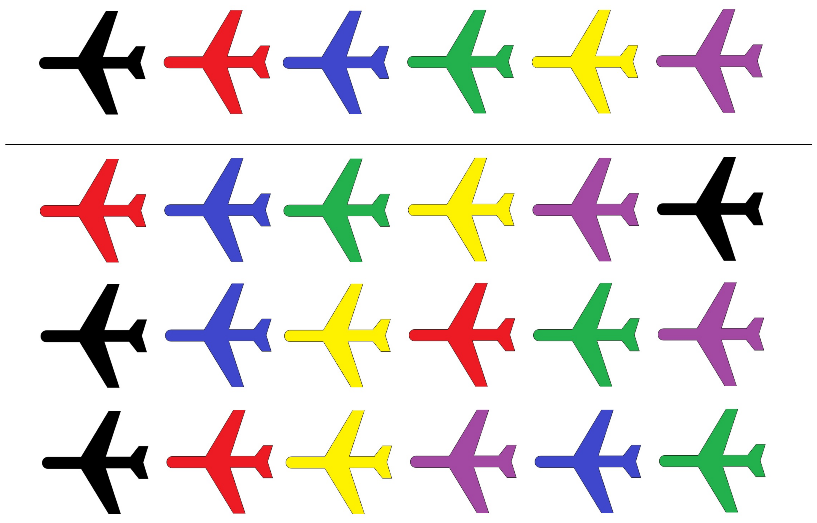


Figure 4.5: Sequence stability illustration

Time Stability

Let P_S^f be the set of airport resources where we want to measure time stability for flight f. By using the definitions from section 3.2.2 we get:

$$\begin{aligned} \zeta_i^f &= \max\left(0, t_i^f - \tau_i^f\right), && i \in P_S^f, f \in F \\ \varepsilon_i^f &= \min\left(0, t_i^f - \tau_i^f\right), && i \in P_S^f, f \in F \end{aligned}$$

which can be rewritten as:

$$\begin{aligned} \zeta_i^f &\geq t_i^f - \tau_i^f, && i \in P_S^f, f \in F \\ \varepsilon_i^f &\leq t_i^f - \tau_i^f, && i \in P_S^f, f \in F \\ \zeta_i^f &\geq 0, && i \in P_S^f, f \in F \\ \varepsilon_i^f &\leq 0, && i \in P_S^f, f \in F \end{aligned} \tag{4.13}$$

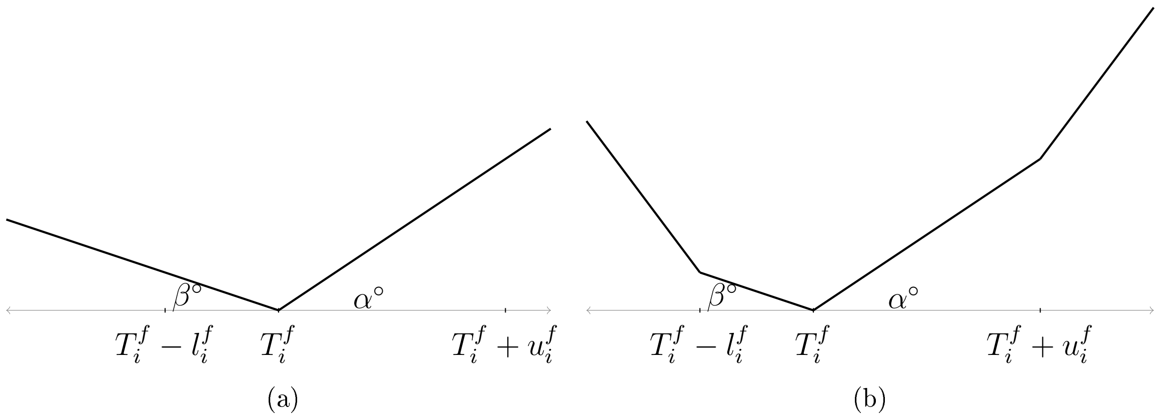
Sequence Stability

Sequence stability was also discussed in section 3.2.2. We will measure sequence stability by counting the number of flights that have a different leading flight on the runway after rescheduling. This measure will not be included in the linear programming model, but it will be used to distinguish between sequence solutions with very similar costs. Since each sequence change need to be communicated, it is better to favour a previously used sequence rather than allow changes with little benefit.

Example 4.2. *Assume we have an initial sequence, s_0 as given above the line in figure 4.5. After an update, we are left with three possibilities s_1, s_2 and with almost the* *same cost. In the first alternative there are only one flight (black) with a new leading flight. The second sequence demands 5 flights with a new leading flight, while the last alternative only demands 2.*

4.4.3 Piecewise linear term

So far we have introduced a convex piecewise linear term for stability (as given in figure 4.6a), and another for punctuality (see figure 4.6b). In section 4.4.2 we introduced costs per time unit if flight f deviated from the target time at a resource i, given by $b_i^{\delta f}$ and $b_i^{\epsilon f}$. If we let $b_i^{\delta f} = 0$ and $b_i^{\epsilon f} = 0$, the term for punctuality will take the form depicted in figure 4.6c. This will give us a window, such that if a flight takes-off or lands inside the window, there are no cost. α and β illustrate that earliness and tardiness can be given different costs (weights).



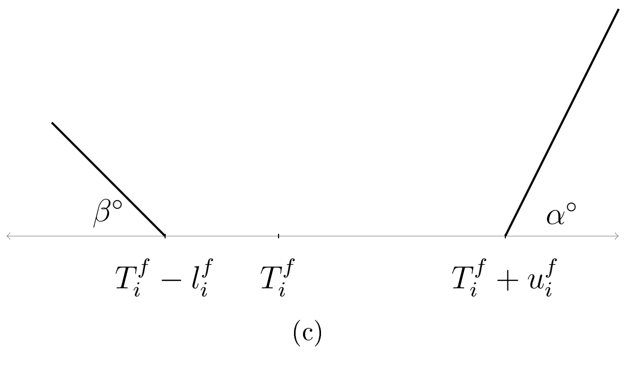


Figure 4.6: Piecewise linear costs

4.5 Summarizing the linear programming model

To end this chapter we will connect all the constraints and the objectives together and present the problem given in step 3 as a linear programming problem. We will also introduce a routegraph, which will be important in the following chapters.

Notation	Definition
D	The set of departing flights
L	The set of landing flights
F	The set of flights, $F = L \cup D$
$R(f)$	The set of legal routes for flight $f \in F$
P_H^f	The set of holding points flight $f \in F$ visits on its route
$P_{\not H}^f$	The set of non-stop points flight $f \in F$ visits on its route, $P_H^f \cap P_{\not H}^f = \emptyset$
P_R^f	The set of airport resources flight $f \in F$ visits on its route, $P_R^f = P_H^f \cup P_{\not H}^f$
P_M^f	The set of milestones for flight $f \in F$, $P_M^f \subseteq P_R^f$
P_S^f	The set of stability points for flight $f \in F$, $P_S^f \subseteq P_R^f$

Table 4.4: Set definitions

Notation	Definition
$c_{i,j}^f$	The driving time for flight f between resource i and resource j
$z_{i,j}^f$	The holding time for flight f between resource i and resource j
$s_i^{f,g}$	The minimum separation between flight f and g at resource i, where flight f goes before flight g
T_i^f	Target time at resource i for flight f
l_i^f, u_i^f	Target window bounds, given the target window $T_i^f \in [T_i^f - l_i^f, T_i^f + u_i^f]$
$b_i^{\delta f}$	The cost per time unit of flight f being late at resource i
$b_i^{\epsilon f}$	The cost per time unit of flight f being early at resource i
$b_{ui}^{\delta f}$	The cost per time unit of flight f being later than u_i^f at resource i
$b_{li}^{\epsilon f}$	The cost per time unit of flight f being earlier than l_i^f at resource i
$b_i^{\zeta f}$	The cost per time unit of changing the time flight f enter resource i
$b_i^{\varepsilon f}$	The cost per time unit of changing the time flight f enter resource i
τ_i^f	The scheduled time for flight f to enter resource i in the previous schedule

Table 4.5: Definitions of parameters, $\{i, j\} \in V, \{f, g\} \in F$

Notation	Definition
δ_i^f	Tardiness for flight f at resource i
δ_{ui}^f	The delay for flight f at resource i leading to a higher cost (delay outside target window)
ϵ_i^f	Earliness for flight f at resource i
ϵ_{li}^f	The earliness for flight f at resource i leading to a higher cost (earliness outside target window)
ζ_i^f	Stability tardiness for flight f at resource i
ε_i^f	Stability earliness for flight f at resource i
t_i^f	The time flight f enter resource i

Table 4.6: Definitions of variables, $i \in V, f \in F$

The problem

Throughout this chapter we have presented constraints and objectives as linear equations and inequalities. If we put all equations together, we get the linear program (4.14).

$$
\begin{array}{lrcll}
\text{minimize} & \multicolumn{4}{l}{\displaystyle\sum_{f\in F}\left(\sum_{i\in P_M^f} b_i^{\delta f}\delta_i^f + b_i^{\epsilon f}\epsilon_i^f + b_{ui}^{\delta f}\delta_{ui}^f + b_{li}^{\epsilon f}\epsilon_{li}^f + \sum_{i\in P_S^f} b_{i\zeta}^f\zeta_i^f + b_{i\varepsilon}^f\varepsilon_i^f\right)} \\
\text{such that[II]} & t_j^f - t_i^f & \geq & c_{i,j}^f, & i\in P_H^f, j\in P_R^f, f\in F \\
 & t_j^f - t_i^f & = & c_{i,j}^f, & i\in P_{\not{H}}^f, j\in P_R^f, f\in F \\
 & t_i^f & \geq & L_i^f, & i\in P_R^f, f\in F \\
 & -t_i^f & \geq & -U_i^f, & i\in P_R^f, f\in F \\
 & t_i^g - t_i^f & \geq & s_i^{f,g}, & i\in P_{\not{H}}^f, i\in P_R^g,\ f,g\in F \\
 & t_i^g - t_j^f & \geq & s_i^{f,g} - c_{i,j}^f, & i\in P_{\not{H}}^f, j\in P_R^f i\in P_R^g,\ f,g\in F \\
 & \delta_i^f - t_i^f & \geq & -T_i^f & i\in P_M^f, f\in F \\
 & \delta_{ui}^f - \delta_i^f & \geq & -u_i^f & i\in P_M^f, f\in F \\
 & t_i^f - \epsilon_i^f & \geq & T_i^f & i\in P_M^f, f\in F \\
 & \epsilon_i^f - \epsilon_{li}^f & \geq & l_i^f & i\in P_M^f, f\in F \\
 & \delta_i^f & \geq & 0, & i\in P_M^f, f\in F \\
 & \delta_{ui}^f & \geq & 0, & i\in P_M^f, f\in F \\
 & -\epsilon_i^f & \geq & 0, & i\in P_M^f, f\in F \\
 & -\epsilon_{li}^f & \geq & 0, & i\in P_M^f, f\in F \\
 & \zeta_i^f - t_i^f & \geq & -\tau_i^f & i\in P_S^f, f\in F \\
 & t_i^f - \varepsilon_i^f & \geq & \tau_i^f & i\in P_S^f, f\in F \\
 & \zeta_i^f & \geq & 0, & i\in P_S^f, f\in F \\
 & -\varepsilon_i^f & \geq & 0, & i\in P_S^f, f\in F
\end{array}
\tag{4.14}
$$

Notice that almost every constraint in (4.14) take the same form. We want to rewrite (4.14) into an equivalent form as the dual of the minimum cost flow problem, since this will allow us to use tools and properties from the network flow theory. Therefore, we want to represent the upper and lower bounds as potential constraints, so we add an extra variable, r, giving us the following model:

[II] assuming flight f goes before flight g, and vertex j is visited immediately after vertex i

$$
\begin{aligned}
\text{minimize} \sum_{f\in F} & \left(\sum_{i\in P_M^f} b_i^{\delta f}\delta_i^f + b_i^{\epsilon f}\epsilon_i^f + b_{ui}^{\delta f}\delta_{ui}^f + b_{li}^{\epsilon f}\epsilon_{li}^f + \sum_{i\in P_S^f} b_{i\zeta}^f\zeta_i^f + b_{i\varepsilon}^f\varepsilon_i^f \right) \\
- \sum_{f\in F} & \left(\sum_{i\in P_M^f\cup P_S^f} (b_i^{\delta f} + b_i^{\epsilon f} + b_{ui}^{\delta f} + b_{li}^{\epsilon f} + b_{i\zeta}^f + b_{i\varepsilon}^f) r \right)
\end{aligned}
$$

$$
\begin{array}{lrcll}
\text{such that} & t_j^f - t_i^f & \geq & c_{i,j}^f, & i\in P_H^f, j\in P_R^f, f\in F \\
& t_j^f - t_i^f & = & c_{i,j}^f, & i\in P_{\cancel{H}}^f, j\in P_R^f, f\in F \\
& t_i^f - r & \geq & L_i^f, & i\in P_R^f, f\in F \\
& r - t_i^f & \geq & -U_i^f, & i\in P_R^f, f\in F \\
& t_i^g - t_i^f & \geq & s_i^{f,g}, & i\in P_{\cancel{H}}^f, i\in P_R^g,\ f,g\in F \\
& t_i^g - t_j^f & \geq & s_i^{f,g} - c_{i,j}^f, & i\in P_{\cancel{H}}^f, j\in P_R^f i\in P_R^g,\ f,g\in F \\
& \delta_i^f - t_i^f & \geq & -T_i^f & i\in P_M^f, f\in F \\
& \delta_{ui}^f - \delta_i^f & \geq & -u_i^f & i\in P_M^f, f\in F \\
& t_i^f - \epsilon_i^f & \geq & T_i^f & i\in P_M^f, f\in F \\
& \epsilon_i^f - \epsilon_{li}^f & \geq & l_i^f & i\in P_M^f, f\in F \\
& \delta_i^f - r & \geq & 0, & i\in P_M^f, f\in F \\
& \delta_{ui}^f - r & \geq & 0, & i\in P_M^f, f\in F \\
& r - \epsilon_i^f & \geq & 0, & i\in P_M^f, f\in F \\
& r - \epsilon_{li}^f & \geq & 0, & i\in P_M^f, f\in F \\
& \zeta_i^f - t_i^f & \geq & -\tau_i^f & i\in P_S^f, f\in F \\
& t_i^f - \varepsilon_i^f & \geq & \tau_i^f & i\in P_S^f, f\in F \\
& \zeta_i^f - r & \geq & 0, & i\in P_S^f, f\in F \\
& r - \varepsilon_i^f & \geq & 0, & i\in P_S^f, f\in F
\end{array}
\qquad (4.15)
$$

Programs (4.14) and (4.15) are equivalent. Every feasible solution $(t', \delta', \epsilon', \zeta', \varepsilon')$ in (4.14) can be transformed into an equal cost feasible solution $(t', \delta', \epsilon', \zeta', \varepsilon', r')$ in (4.15) by letting $r' = 0$. In the same way, every feasible solution $(t', \delta', \epsilon', \zeta', \varepsilon', r')$ to (4.15) can be converted to an equal cost feasible solution $(\bar{t}, \bar{\delta}, \bar{\epsilon}, \bar{\zeta}, \bar{\varepsilon})$ to (4.14) by letting $\bar{t} = t' - r', \bar{\delta} = \delta' - r', \bar{\epsilon} = \epsilon' - r', \bar{\zeta} = \zeta' - r', \bar{\varepsilon} = \varepsilon' - r'$.

Notice that all the constraints given in (4.15) are written in the same form, and to write this problem as the dual of the minimum cost flow problem, we only need to change all the signs:

$$
\begin{aligned}
\text{maximize} - \sum_{f \in F} & \left(\sum_{i \in P_M^f} b_i^{\delta f} \delta_i^f + b_i^{\epsilon f} \epsilon_i^f + b_{ui}^{\delta f} \delta_{ui}^f + b_{li}^{\epsilon f} \epsilon_{li}^f + \sum_{i \in P_S^f} b_{i\zeta}^f \zeta_i^f + b_{i\varepsilon}^f \varepsilon_i^f \right) \\
& + \sum_{f \in F} \left(\sum_{i \in P_M^f \cup P_S^f} (b_i^{\delta f} + b_i^{\epsilon f} + b_{ui}^{\delta f} + b_{li}^{\epsilon f} + b_{i\zeta}^f + b_{i\varepsilon}^f) r \right)
\end{aligned}
$$

$$
\begin{array}{rrll}
\text{such that} & t_i^f - t_j^f \leq & -c_{i,j}^f, & i \in P_H^f, j \in P_R^f, f \in F \\
& t_j^f - t_i^f \leq & c_{i,j}^f, & i \in P_{\not{H}}^f, j \in P_R^f, f \in F \\
& t_i^f - t_j^f \leq & -c_{i,j}^f, & i \in P_{\not{H}}^f, j \in P_R^f, f \in F \\
& r - t_i^f \leq & -L_i^f, & i \in P_R^f, f \in F \\
& t_i^f - r \leq & U_i^f, & i \in P_R^f, f \in F \\
& t_i^f - t_i^g \leq & -s_i^{f,g}, & i \in P_{\not{H}}^f, i \in P_R^g,\ f, g \in F \\
& t_j^f - t_i^g \leq & c_{i,j}^f - s_i^{f,g}, & i \in P_{\not{H}}^f, j \in P_R^f i \in P_R^g,\ f, g \in F \\
& t_i^f - \delta_i^f \leq & T_i^f & i \in P_M^f, f \in F \\
& \delta_i^f - \delta_{ui}^f \leq & u_i^f & i \in P_M^f, f \in F \\
& \epsilon_i^f - t_i^f \leq & -T_i^f & i \in P_M^f, f \in F \\
& \epsilon_{li}^f - \epsilon_i^f \leq & -l_i^f & i \in P_M^f, f \in F \\
& r - \delta_i^f \leq & 0, & i \in P_M^f, f \in F \\
& r - \delta_{ui}^f \leq & 0, & i \in P_M^f, f \in F \\
& \epsilon_i^f - r \leq & 0, & i \in P_M^f, f \in F \\
& \epsilon_{li}^f - r \leq & 0, & i \in P_M^f, f \in F \\
& t_i^f - \zeta_i^f \leq & \tau_i^f & i \in P_S^f, f \in F \\
& \varepsilon_i^f - t_i^f \leq & -\tau_i^f & i \in P_S^f, f \in F \\
& r - \zeta_i^f \leq & 0, & i \in P_S^f, f \in F \\
& \varepsilon_i^f - r \leq & 0, & i \in P_S^f, f \in F
\end{array}
\tag{4.16}
$$

4.5.1 Constructing a routegraph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$

Next, we associate in a standard fashion flow network $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ with the linear program (4.16) above, which we have constructed as the dual of a minimum cost flow problem. In section 2.3 we defined this problem as follows:

$$
\begin{array}{lll}
\text{maximize} & \sum_{i \in V} b_i y_i & \\
\text{s.t} & y_j - y_i \leq c_{i,j}, & (i, j) \in E
\end{array}
$$

We also noted that network flow problems are defined on graphs, where $c_{i,j}$ is the edge cost of the edge (i, j) and b_i is the amount vertex i supply. Therefore, each variable in (4.16) gets associated a vertex v. With each vertex we associate a supply, which equals the coefficient of the associated variable in the objective function, giving us the following cases:

- Original vertices: v is associated with variable t_i^f. There is no associated coefficient in the objective function, so its supply will be 0.

- Delay vertices: v is associated with the lateness variable δ_i^f and the associated coefficient is $b_i^{\delta f}$.
- Early vertices: v is associated with the earliness variable ϵ_i^f and the associated coefficient is $b_i^{\epsilon f}$.
- Additional delay vertices: v is associated with the lateness variable δ_{ui}^f and the associated coefficient $b_{ui}^{\delta f}$.
- Additional early vertices: v is associated with the earliness variable ϵ_{li}^f and the associated coefficient $b_{li}^{\epsilon f}$.
- Tardiness stability vertices: v is associated with the stability lateness variable ζ_i^f and the associated coefficient is $b_i^{\zeta f}$.
- Earliness stability vertices: v is associated with the stability earliness variable ε_i^f and the associated coefficient is $b_i^{\varepsilon f}$.
- Root vertex: v is associated with the variable r, its supply will be

$$-\left(\sum_{i\in P_M^f} b_i^{\delta f} + \sum_{i\in P_M^f} b_i^{\epsilon f} + \sum_{i\in P_M^f} b_{ui}^{\delta f} + \sum_{i\in P_M^f} b_{li}^{\epsilon f} + \sum_{i\in P_S^f} b_i^{\zeta f} + \sum_{i\in P_S^f} b_i^{\varepsilon f}\right)$$

In the same way, for each constraint $v - u \leq C$ in (4.16) we associate a directed edge $(u, v) \in \mathscr{E}$ with edge cost C. The edges in the network can be classified as follows:

- Original edges: For each constraint $t_j^f - t_i^f \leq -c_{i,j}^f$, let u be the vertex representing flight f entering resource i and v be the vertex representing flight f entering resource j. Then there is an edge (u, v) in $\mathscr{E}$ with cost $-c_{i,j}^f$. Similar, for $t_i^f - t_j^f \leq c_{i,j}^f$ there is an edge (v, u) with edge cost $c_{i,j}^f$.
- Separation edges: For each constraint $t_i^f - t_i^g \leq -s_i^{f,g}$ let u be the vertex representing flight f entering resource i and v be the vertex representing flight g entering resource i. Then there is an edge (v, u) in $\mathscr{E}$ with cost $-s_i^{f,g}$. When i is a holding vertex for flight f, $t_j^f - t_i^g \leq c_{i,j}^f - s_i^{f,g}$ will give us an edge from the vertex representing flight g entering i to the vertex representing flight f entering j with edgecost $c_{i,j}^f - s_i^{f,g}$.
- Delay edges: For each constraint $t_i^f - \delta_i^f \leq T_i^f$ we add a delay edge. If u is the delay vertex associated with the variable δ_i^f, and v is the original vertex associated with the variable t_i^f, then there is an edge $(u, v) \in \mathscr{E}$ with cost T_i^f, where T_i^f is the target time for flight f at point i in the airport graph. For each constraint $\delta_i^f - \delta_{ui}^f \leq u_i^f$ there is an edge from the additional delay vertex representing δ_{ui}^f to the delay vertex representing δ_i^f with edgecost u_i^f.
- Early edges: For each constraint $\epsilon_i^f - t_i^f \leq -T_i^f$ we add an early edge. If u is the early vertex associated with the variable ϵ_i^f, and v is the original vertex associated with the variable t_i^f, then there is an edge $(v, u) \in \mathscr{E}$ with cost $-T_i^f$, where T_i^f is the target time for flight f at point i in the airport graph. For each constraint $\epsilon_{li}^f - \epsilon_i^f \leq -l_i^f$ there is an edge from the early vertex representing ϵ_i^f to the additional early vertex representing ϵ_{li}^f with edgecost $-l_i^f$.

- Lower bound edges: For each constraint $r - t_i^f \leq -L_i^f$ we associate u with the variable t_i^f and v with r, then there is an edge $(u, v) \in \mathscr{E}$ with cost $-L_i^f$.
- Upper bound edges: For each constraint $t_i^f - r \leq U_i^f$ we associate u with the variable t_i^f and v with r, then there is an edge $(v, u) \in \mathscr{E}$ with cost U_i^f.
- Late stability edges: For each constraint $t_i^f - \zeta_i^f \leq \tau_i^f$ we add a late stability edge. If u is the tardiness stability vertex associated with the variable ζ_i^f, and v is the original vertex associated with the variable t_i^f, then there is an edge $(u, v) \in \mathscr{E}$ with cost τ_i^f, where τ_i^f is the time flight f was calculated to arrive at point i in the last schedule.
- Early stability edges: For each constraint $\varepsilon_i^f - t_i^f \leq -\tau_i^f$ we add an early stability edge. If u is the earliness stability vertex associated with the variable ε_i^f, and v is the original vertex associated with the variable t_i^f, then there is an edge $(v, u) \in \mathscr{E}$ with cost $-\tau_i^f$, where τ_i^f is the time flight f was calculated to arrive at point i in the last schedule.
- Positive delay/positive tardiness stability: if u is a delay or tardiness stability vertex (associated with the variable $\delta_i^f, \delta_{ui}^f$ or ζ_i^f), then there is an edge $(u, r) \in \mathscr{E}$ with cost 0.
- Negative early/negative earliness stability: if u is an early or earliness stability vertex (associated with the variable $\epsilon_i^f, \epsilon_{li}^f$ or ε_i^f), then there is an edge $(r, u) \in \mathscr{E}$ with cost 0.

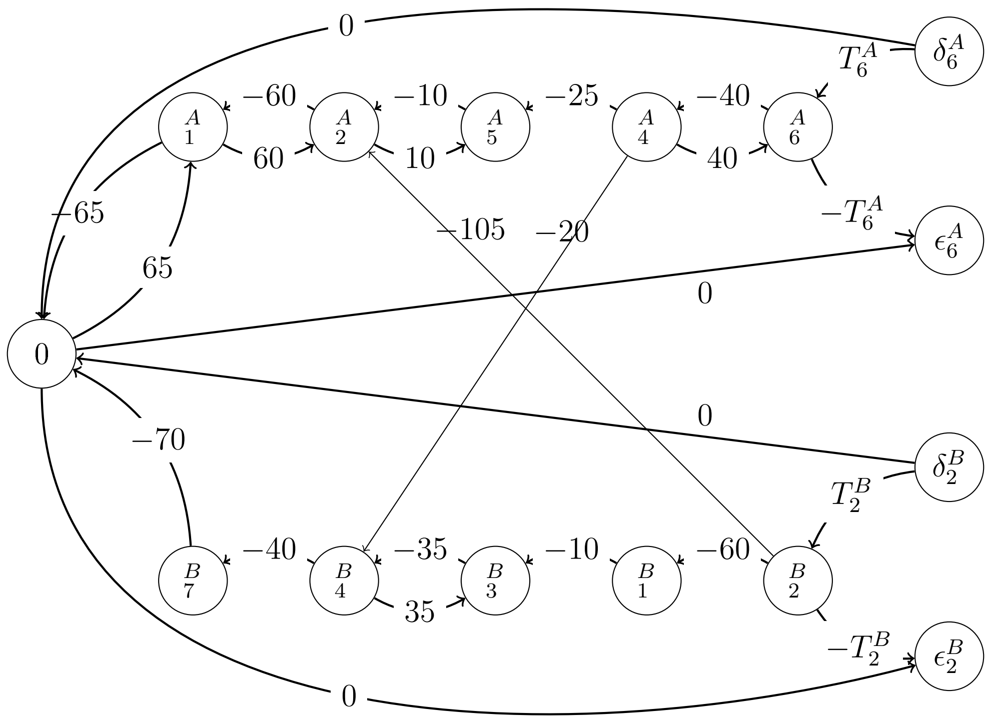


Figure 4.7: Routegraph - Example 4.1

Routegraph example

Example 4.1 (continued) *The complete linear program for this example is given by:*

$$\text{minimize} \qquad \sum_{f\in F}\left(\sum_{i\in P_D} b^f_{i\delta}\delta^f_i + b^f_{i\epsilon}\epsilon^f_i - \sum_{i\in P_D}(b^f_{i\delta}+b^f_{i\epsilon})r\right)$$

$$\begin{aligned}
t^A_2 - t^A_1 &= 60 & \qquad t^B_7 - t^A_4 &\le -40\\
t^A_5 - t^A_2 &\le 10 & t^A_3 - t^B_4 &= 35\\
t^A_5 - t^A_4 &\le -25 & t^B_3 - t^B_1 &\le -10\\
t^A_6 - t^A_4 &= 40 & t^B_1 - t^B_2 &\le -60\\
t^A_1 - r &= 65 & r - t^B_7 &\le -70\\
\epsilon^A_6 - t^A_6 &\le -T^A_6 & \epsilon^B_2 - t^B_2 &\le -T^B_2\\
t^A_6 - \delta^A_6 &\le T^A_6 & t^B_2 - \delta^B_2 &\le T^B_2\\
\epsilon^A_6 - r &\le 0 & \epsilon^B_2 - r &\le 0\\
r - \delta^A_6 &\le 0 & r - \delta^B_2 &\le 0\\
t^B_4 - t^A_4 &\le -20 & t^A_2 - t^B_2 &\le -105
\end{aligned}$$

Applying the instructions on page 40 will give us the routegraph in figure 4.7.

Illustration of separation constraints

We are now ready to illustrate the separation constraints on the runway, as given in table 4.3. As noted in section 4.3.2, the separation constraints will vary depending on the sequence. Here we will look at two flight, A and B, where we assume that flight A will use the runway before flight B. In total, we need to visualize four vertices for each flight: three representing positions on the runway (runway entry, the second runway position and runway exit) and the first taxiway position after the runway. Some vertices appear in all four cases (e.g. the runway exit for flight B), while other only appear in one situation (e.g. the runway entry for flight A). To make it easier to compare the four cases, only the separation edges are illustrated. In addition, only three vertices will represent the runway.

In figure 4.8 we look at the separation constraint when there is two arriving flights after each other. From table 4.3 this separation is given by $t^A_T - t^B_E \le c^A_{E,T} - s^{A,B}_E$, and is represented by an edge in the routegraph. This edge has a cost of $c_{E,T} - s^{A,B}_E$. The same idea lies behind figure 4.9 - figure 4.11.

In figure 4.9 we look at the separation constraints when there is a departing flight after an arriving, and in figure 4.10 when there is a departing flight before an arriving. The final figure (figure 4.11) represent the separation constrains that is active when there are two departing flights after each other. Notice that there are two separation constraints when the second flight is a departing flight.

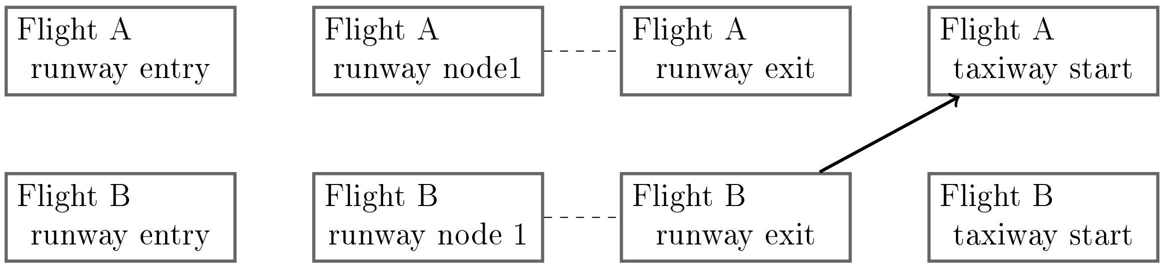


Figure 4.8: Runway Separation: Arrival - Arrival

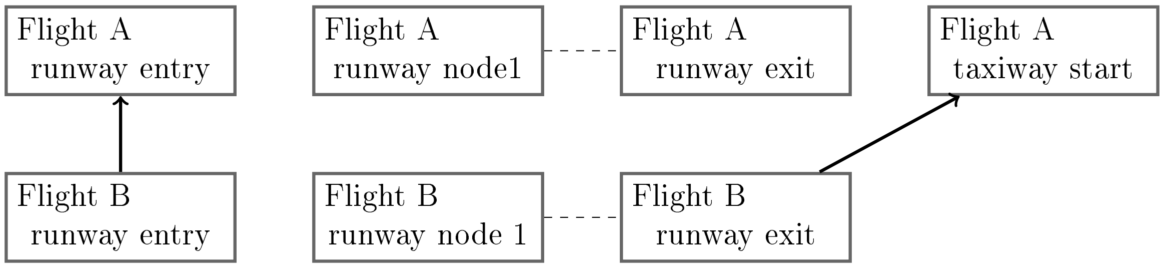


Figure 4.9: Runway Separation: Arrival - Departure

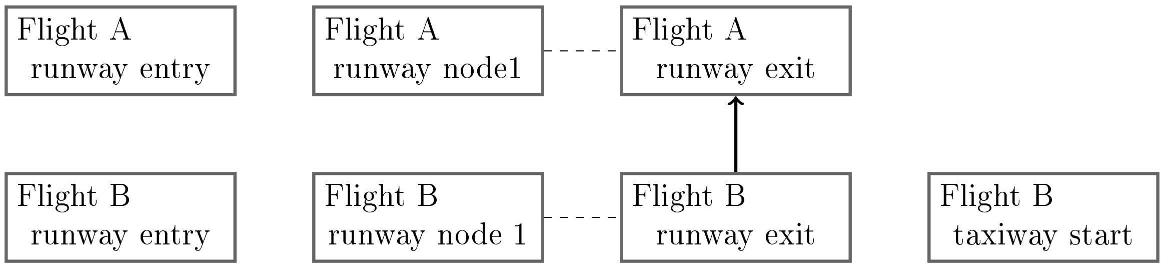


Figure 4.10: Runway Separation: Departure - Arrival

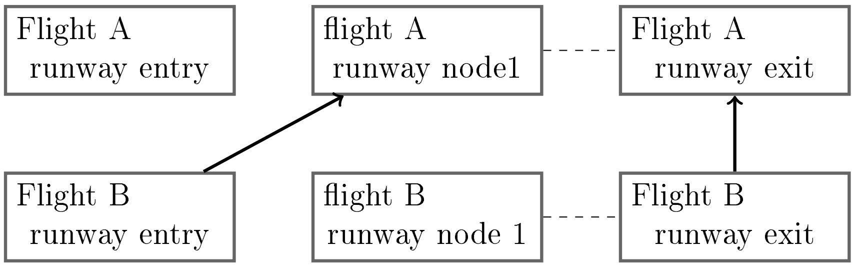


Figure 4.11: Runway Separation: Departure - Departure

Chapter 5

Algorithm and implementation details

In this chapter we will present some implementation details and prepare for the test runs in chapter 6. The implementation in this thesis was done using using $C\#$.

In Chapter 4 we outlined four solution steps for our main problem:

1. Find for all $f \in F$ the shortest legal route $r^f \in R(f)$
2. For all non-shareable resources, find a precedence sequence and associated precedence constraints
3. Compute a conflict-free solution, respecting the context established in step 1 and 2
4. If new information appear, revise the schedule

We will look at the first step, the problem of finding the shortest legal route for a flight, in the first section. Then we will discuss how to find a precedence sequence (step 2) and how to solve any potential conflicts. In chapter 4 we described the problem in step three as the dual of the minimum cost flow problem. In section 5.3 we will focus on finding an initial tree solution to this problem. In section 5.4 we will discuss some issues and approaches related to rescheduling (step 4) and in section 5.5 we look at feasibility and negative cycles. We will end this chapter presenting the main algorithm for solving the overall scheduling problem, and discuss the performance.

5.1 Shortest legal route

As discussed earlier, each flight $f \in F$ is assigned a taxi route between the runway and a gate in the second solution step, see figure 4.3 at page 26. To be feasible, the taxi route should take the taxi route procedures into account. These are presented and illustrated in appendix A.

In order to quickly find the shortest feasible route for each flight, the airport graph presented in section 4.2 contains information about the direction each segment can be used. One-way streets are represented with directed segments which can only be traversed in one direction and undirected segments can be traversed in both directions. In figure 5.1

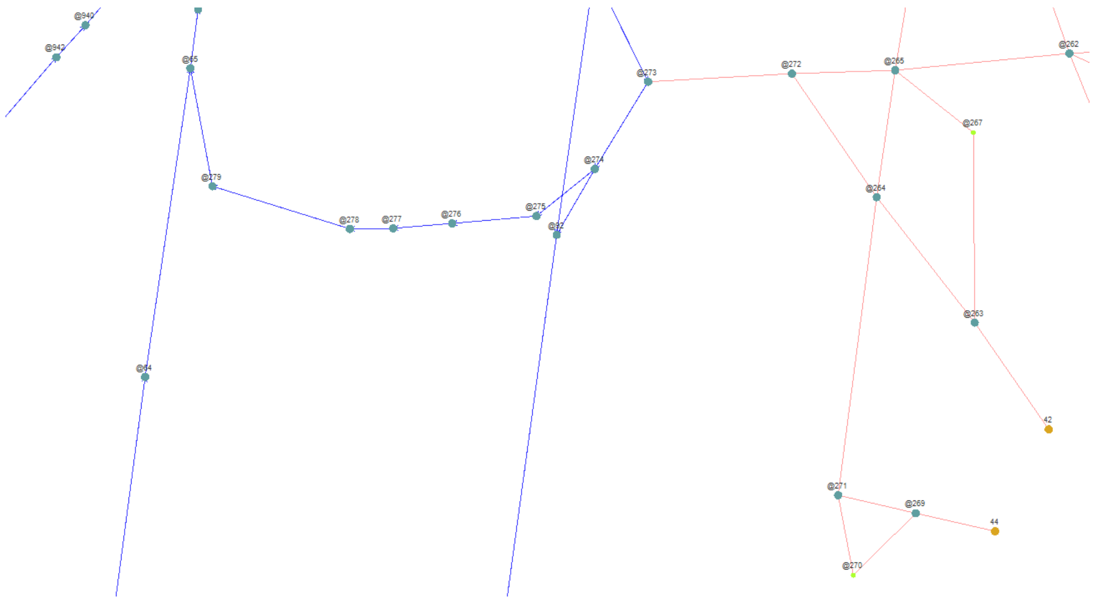


Figure 5.1: Airport graph of the manoeuvring area outside terminal 4

we zoom in on a particular area of the airport graph (figure 4.1) to get a better view. This area is a part of the manoeuvring area outside terminal 4 at Arlanda, and gate 42 and 44 are marked with two orange dots in the bottom right corner. Here the undirected segments are drawn in orange, while the directed segments are drawn as blue arrows.

Using the interpretation that a movement at a directed edge is only permitted from the start vertex to the end vertex, whereas movements are permitted in both directions at an undirected edge, any shortest path algorithm can be applied to the airport graph to find the shortest feasible route for a flight f.

5.2 Non-shareable resources

For most of the airport resources, only one aircraft can use it at time, so a precedence decision on "who goes first" need to be taken. For the experiments we wanted a complete search algorithm, to guarantee to find the bes possible solution. Therefore, a simple sequencing algorithm we chosen and we enumerate all possible sequences on the runway. Given a list of flights that will use an airport resource i, the algorithm given in listings 5.1 will return all possible permutations. This is time-consuming, but by enumerating all possible precedence sequences, we are guaranteed to find an optimal solution to the main problem.

The idea behind this, was that we want to compare the optimal solution, when only considering punctuality, against the optimal solutions, when also considering stability. Using a heuristic, we are to able to find good solutions, but we have no guarantee that we get an optimal solution.

When the number of flights are high, enumerating all possible solutions is not executable since the growth rate are factorial.

An alternative approach to finding all precedence sequences would be to find a solution that satisfies the taxi route constraints presented in section 4.3.1. Then, search through the schedule vectors and look for potential conflicts, that is two or more flights use the same resource simultaneously. Solve the conflict by choosing "who goes first" after some given rules, for example a departure goes before an arrival, add a constraint and then recompute. See section 5.4 for a further discussion of adding constraints.

Listing 5.1: Complete seach algorithm for finding all possible precedence sequences

```
/// <summary>
/// This function takes a list of integers as input and returns
/// all possible permutations.
/// e.g.
/// Input:      List with integers 1,2 and 3
/// Output:     A new list with elements: (1,2,3), (1,3,2),
///             (2,1,3), (2,3,1), (3,1,2)
/// </summary>
public static IEnumerable<List<int>> Permutations(List<int>list)
        {
            if (list.Count == 1)
            {
                yield return list;
                yield break;
            }

            for (int index = 0; index < list.Count; ++index)
            {
                int element = list[index];
                List<int> rest = new List<int>(list);
                rest.RemoveAt(index);

                foreach (var perm in Permutations(rest))
                {
                    perm.Insert(0, element);
                    yield return perm;
                }
            }
        }
```

5.3 Initial tree

After generating a routegraph, as described in section 4.5.1, we want to find an initial spanning tree. There are a number of algorithms available for finding a tree solution, but since our problem is given as a dual network simplex problem, we would like to start with a dual feasible solution.

5.3.1 Bellman-Ford Steps

The Bellman-Ford algorithm, presented in chapter 2, is a method that computes the shortest paths from a single root vertex, to all other vertices in the graph. In this section we will use the same notation and assumption as given in Algorithm 3 at page 12.

Definition 5.1. A shortest paths tree is a directed subgraph $G' = (V', E')$ of $G = (V, E)$, where $V' \subseteq V$ and $E' \subseteq E$, such that

1. V' is the set of vertices reachable from a source s in G,
2. G' form a rooted tree with root s, and
3. for all $v \in V'$, the unique simple path from s to v in G' is a shortest path from s to v in G.

Lemma 5.2. *Given a connected graph $G = (V, E)$ with edgecosts $c_{i,j}$, such that the Bellman-Ford algorithm terminates with a minimum path length $y_i < \infty$ for all $i \in V$, then the associated tree is dual feasible.*

Proof. If the Bellman-Ford algorithm terminates with a length $y_i < \infty$ for all vertices $i \in V$, then $y_j - y_i \leq c_{i,j}$ is satisfied for all $(i, j) \in E$, hence the associated tree is dual feasible. □

The problem is that there are often some vertices in the graph that are not reachable from the root, and since all lengths are set by default to infinity in the beginning of the Bellman-Ford algorithm (Algorithm 3), it will return infinity as value on the shortest path between the root and these vertices. In figure 5.2 at page 52 we see that vertex b and c are reachable from a, but d and e are not. In this, and in similar cases, the Bellman-Ford algorithm will not give us an associated spanning tree. However, $y_j - y_i \leq c_{i,j}$ is satisfied for all vertices reachable from root vertex r. Using Bellman-Ford repeatedly, each time with a different root vertex and only visiting vertices that have not yet been visited, will create subtrees G' of G, and each subtree satisfy $y_j - y_i \leq c_{i,j}$ for all $i, j \in V'$

If we are able to reconnect every subtree G' of G such that $y_j - y_i \leq c_{i,j}$ also is satisfied in the cut between each pair of subtrees, then the corresponding tree solution will be dual feasible.

Theorem 5.3. *Given a connected graph $G = (V, E)$ with edgecosts $c_{i,j}$, the initial dual solution algorithm (Algorithm 7) will terminate with a dual feasible spanning tree, if such exist.*

Proof. **Case 1: all vertices reachable from a root vertex r:** If all vertices are reachable from r, Algorithm 7 equals the Bellman-Ford algorithm, and from Lemma 5.2 we know that this algorithm terminates with a dual feasible spanning tree.
Case 2: not all vertices are reachable from a root vertex r: Let V_1 be he vertices reachable from r and let V_2 be the vertices reachable from r', where r' is the new root vertex (the first vertex not reachable from r). For simplicity, we assume that $V_1 \cup V_2 = V$.

Algorithm 7 Initial Dual Solution Algorithm

```
Input: Connected Directed graph $G = (V, E)$, Edgecost $c_{i,j}$ and a root vertex r
  1.   for every vertex v
           $y_v = \infty$;
           $visited_v = false$
       end for
       go to step 2
  2.   $y_r = 0$;
       let $V_1$ be the vertices reachable from r
       for $i \in V_1$
           find shortest path $p$ from $r$ to $i$ using the Bellman-Ford algorithm. The
           length of the path is given by $y_i$
       end for
       go to step 3
  3.   if $V_1 = V$
           return $y_i$, $i \in V$
       else
           let $j$ be the first vertex not visited ($visited_j = false$), s.t. there is at least
           one edge from $j$ to $V_1$, and let $V_2$ be the vertices reachable from $j$, not yet
           visited
           $r' = j$;
           go to step 4
       end if
  4.   $y_{r'} = 0$;
       for $i' \in V_2$
           find shortest path $y_{i'}$ from $r'$ to $i'$ using the Bellman-Ford algorithm
       end for
       $\Delta = \min(c_{u,v} - y_v + y_{u'}), u \in V_2, v \in V_1$
       for each $i \in V_2$
           $y_i = y_{i'} - \Delta$;
       end for
       $V_1 = V_1 \cup V_2$
       go to step 3
```

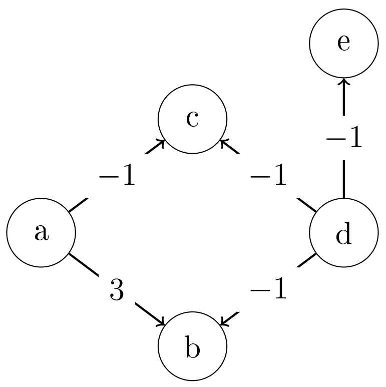


Figure 5.2: Connected directed graph

Running Bellman-Ford twice will create two subtrees with the property that:

$$y_j - y_i \leq c_{i,j}, \qquad i, j \in V_1$$

and

$$y_v - y_u \leq c_{u,v}, \qquad u, v \in V_2.$$

Since G is connected, there is at least one edge in the cut between V_1 and V_2. Let $E' \subset E$ denote the set of these edges. If we can prove that $y_j - y_i \leq c_{i,j}$ for $(i, j) \in E'$, then the initial dual solution algorithm will terminate with a dual feasible solution.

Depending on the values in the two subtrees, the edges in the cut may lead to the following equations:

$$y_v - y_u = c_{u,v}, \qquad u \in V_2 \text{ and } v \in V_1, \tag{5.1}$$
$$y_v - y_u < c_{u,v}, \qquad u \in V_2 \text{ and } v \in V_1, \tag{5.2}$$
$$y_v - y_u > c_{u,v}, \qquad u \in V_2 \text{ and } v \in V_1. \tag{5.3}$$

Since $y_v - y_u \leq c_{u,v}$ is equivalent to $(y_v + \Delta) - (y_u + \Delta) \leq c_{u,v}$, we can adjust the variables in the second subtree with Δ, and the new variables, $y'_u = y_u + \Delta$, will still satisfy the dual feasibility equation for vertices in V_2.

Let $\Delta_u = c_{u,v} - y_v + y_u$ represent the distance between the left and the right hand side of the equations above. Then $\Delta_u = 0$ for equation 5.1, $\Delta_u > 0$ for equation 5.2 and $\Delta_u < 0$ for equation 5.3. Let $\Delta = \min\limits_u \Delta_u$, and update the dual variables in V_2 as follows:

$$y'_u = y_u + \Delta, \qquad u \in V_2.$$

Then $y_v - y_u \leq c_{u,v}$ for $(u, v) \in E'$, with at least one edge satisfying with equality. If there are more than one equation at equality, choose one of them to enter the tree.

□

Remark 1. If there are some vertices that have not been visited after step 4, repeat step 3 and 4. The proof will be analog to what has been proven above.

Example 5.4. *Let $G = (V, E)$ be given as figure 5.2, with edgecost as given in the graph. Running Algorithm 7 with a as root vertex, the algorithm will return $y_a = 0, y_b = 3, y_c = -1$, and vertex d and e are not visited after step 1. In step 2 we set $R_{new} = d$, giving $y_d = 0, y_e = 2]$. Now, $V_1 = \{a, b, c\}$ and $V_2 = \{d, e\}$, and in the cut between the two subsets, we find two edges, (d, b) and (d, c). The $\Delta-$variables for the two edges:*

- $\Delta_{d,b} = c_{d,b} - y_b + y_d = -1 - 3 - 0 = -4$
- $\Delta_{d,c} = c_{d,c} - y_b + y_d = 2 - (-1) - 0 = 3$

This gives us $\Delta = -4$, so $y_d = 0 - \Delta = 4$, $y_e = 2 - \Delta = 6$, such that $y_c - y_d = -1 - 4 = -5 < 2 = c_{d,c}$ and $y_b - y_d = 3 - 4 = -1 = c_{d,b}$.

Remark 2. If e had been chosen as the new root vertex instead of d in the example above, then $V_1 = \{a, b, c\}$ and $V_2 = \{e\}$. Since there are no edges in the cut between V_1 and V_2, we would not be able to connect these subsets in one operation. This could be solved by creating a third subtree, $V_3 = \{d\}$, and then connect V_1 and V_3 first. This is why step 3 in Algorithm 7 demands that there is an edge from the root vertex in V_2 to a vertex in V_1.

5.4 Rescheduling

Another important question in the scheduling process, is how to reschedule, and this is the fourth solution step for our main problem. As discussed briefly in section 3.2.2, any disruption may provoke the need of revising the initial generated schedule. In this section we will assume that an initial schedule has been generated, and we will refer to this as the original schedule or original dual solution. Since the scheduling problem from the thirds solution step is the dual of the minimum cost flow problem, there is also an original primal solution associated with this problem. Here we will look at some disruptions and how the schedule can be revised to assure that the realized schedule is feasible.

Change a constraint-value

The first change we will consider, is when the controller receive updated information about the flight movements. If a flight is delayed from gate, has received a new landing time or a new target take-off time, we need to update the right-hand side of the corresponding constraints in the problem formulation (4.16).

By the way we constructed the routegraph in section 4.5.1, a change of the value of the right hand side, will result in a change of the edgecost of the corresponding edge in the routegraph, meaning that the original schedule may not longer be feasible. However, since the supply at each vertex stays unchanged, the original primal solution is still feasible, and this solution can be used as an initial primal solution in the rescheduling problem.

Additional constraints

Another change in the problem formulation is when new information leads to additional constraints. When we add a new constraint, we also need to add an edge in our route-graph. Since adding a new edge does not change the supply of any vertex, the original primal solution will still be primal feasible in the rescheduling problem.

Remove flight from schedule

When a flight is cancelled or does not make any manoeuvre during the time horizon H, it can be removed from the scheduling problem. When we remove a flight, we simply remove all vertices and edges corresponding to this flight. In addition, we might need to update the supply at the root vertex, r, such that the total supply still sum up to zero. Changing the supply at r will change the primal problem constraints, but the dual problem will still maintain the same, so we can use a part of the original dual solution as the initial solution in the rescheduling problem.

Add flight to schedule

When a new flight is added to the schedule, we need to introduce new variables and new constraints. As in the case where vertices are removed, we may need to update the supply at the root vertex, r.

The original dual solution, corresponds to a dual feasible tree for this part of the rescheduling problem, and it is not necessary to recompute this part. If we let V_1 be the vertices in this tree, and say that all the new vertices are unvisited, we can use the initial dual solution algorithm (Algorithm 7), starting at step 3. As proven earlier in this chapter, the initial dual solution algorithm, terminates with a dual feasible tree solution, which can be used as an initial solution to our rescheduling problem.

Reorder sequence

In most of the cases discussed above, it is not necessary to reorder the sequence to get a good solution. However, in some cases a resequencing might give a more efficient schedule.

Since changing the sequence, definitely means that the separation constraint are changed, we cannot use the original dual solution, since this is no longer feasible. Whether we can use the original primal solution as an initial solution in the rescheduling problem, will depend on whether the separation edge we want to remove appear in the optimal solution tree or not. If the edge does not appear in the tree, the flow on this edge is zero by the complementary slackness theorem, and we can remove this edge and still have a tree solution.

The same is true if the separation edge appear in the optimal tree solution, but have a flow equal to zero. Then we can remove this edge and replace it with another edge with flow equal to zero, such that we still have a tree solution.

On the other hand, if the separation edge appear in the optimal solution tree and the flow on this edge is greater than zero, we cannot remove this edge and replace it with an edge in the opposite direction because this will lead to a negative flow on the new edge, and this tree will not be primal feasible. In this case we need to compute a new initial solution.

5.5 Feasibility

From linear programming theory, we know that an LP problem can be either feasible or infeasible. In this section we will explore when our problem is (dual) infeasible.

Primal \ Dual	Optimal	Unbounded	Infeasible
Optimal	✓	–	–
Unbounded	–	–	✓
Infeasible	–	✓	✓

Table 5.1: Feasibility and Unboundedness

5.5.1 Negative cycle

A negative cycle is a directed cycle whose edges sum up to a negative value. If a graph contains a negative cycle, then the corresponding constraints are inconsistent, and no feasible solution exist. The problem is infeasible.

Example 5.5. *In figure 5.3 and in figure 5.4 there are given two examples of a runway sequence between two arriving flights at runway 19R at Arlanda. Runway 19R is represented by 8 vertices (12, 19, 129, 128, 121, 109, 861, 42) in the Arlanda airport graph, illustrated by a chart in figure 4.1 at page 24. We let the scheduling horizon begin at 07:00:00, and the first flight, A, has a Target Time of Arrival at 07:22:00 and the second flight, B, at 07:25:00. Both flights follow the same landing procedure, and will use the same amount of time on the runway. We assume that an arriving flight can land* 1.5 *minute before or after the target time. This means that flight A can land between 1230 and 1410 and flight B between 1410 and 1590. We are now going to consider two possible sequences on the runway.*

In figure 5.3 we look at a sequence where flight B will try to land before flight A. If we sum up the edgecosts in the highlighted cycle, we get a negative value of -80*, meaning that this sequence is not allowed. There is no way that flight B can land before flight A.*

In figure 5.4 we illustrate that flight A land before flight B. Here, there are no negative cycles, meaning that this sequence is allowed, and flight A can land before flight B.

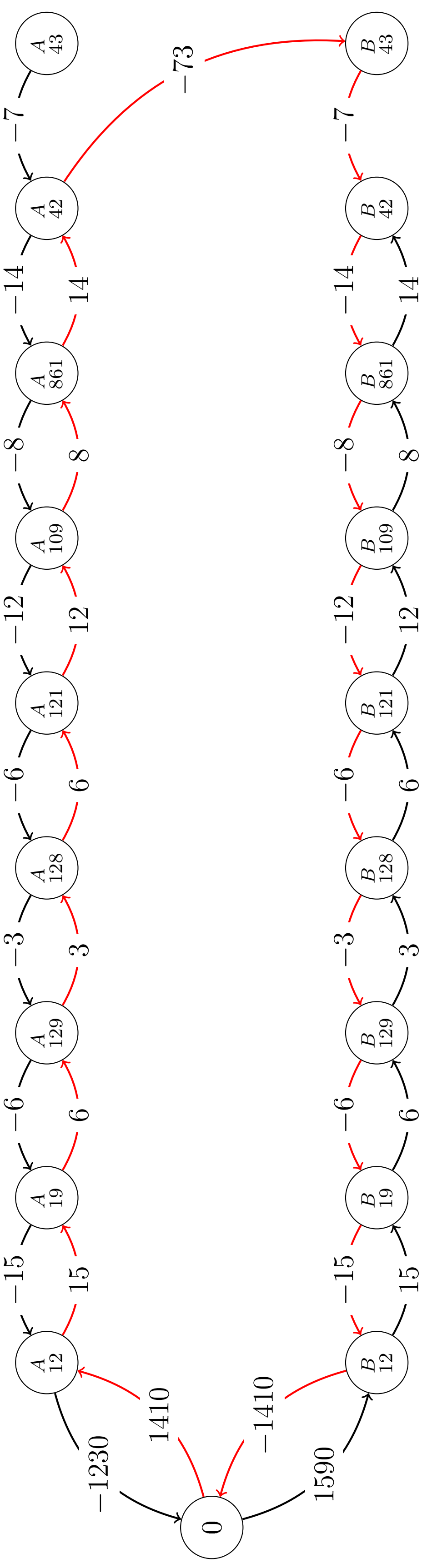


Figure 5.3: Negative cycle in Routegraph - Two Planes

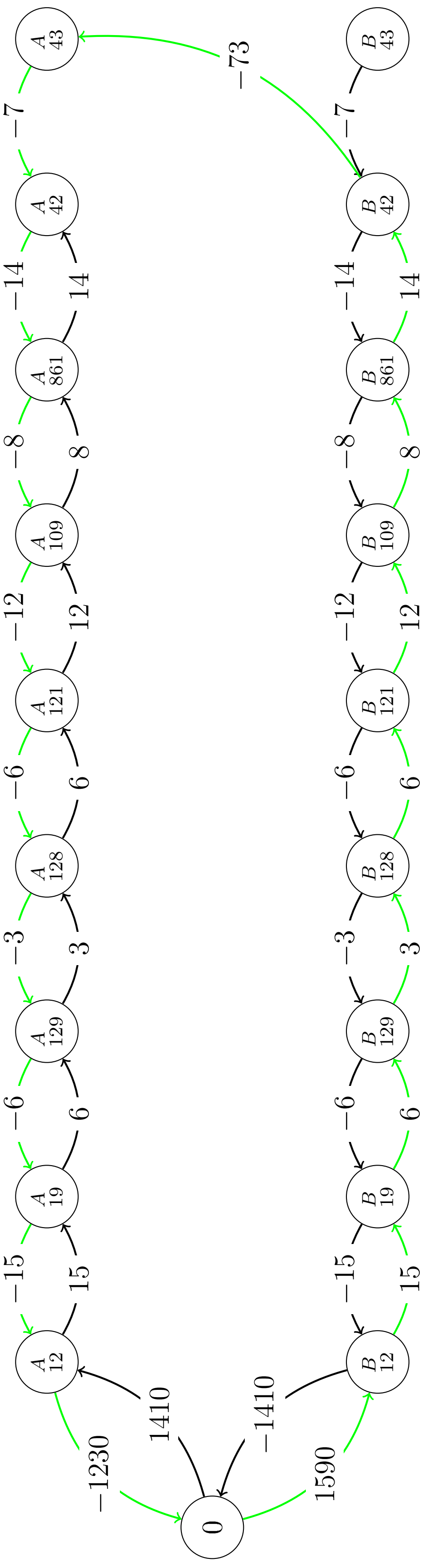


Figure 5.4: Nonnegative cycle in Routegraph - Two Planes

5.6 Main Algorithm

We started this chapter by repeating the four solution steps for our problem. The first step was to find the shortest legal route for each flight, this was discussed in section 5.1. The second step was to find all non-shareable resources and generate precedence sequences for using these. This was the topic of section 5.2. The third step was to find a conflict-free solution, respecting the context given from the two first steps. This was mainly discussed in Chapter 4, and we presented an algorithm for finding an initial solution in section 5.3. The final step, rescheduling, was discussed in section 5.4. Together, they give Algorithm 8.

Algorithm 8 Main Algorithm

```
Input: Connected airport graph $G = (V, E)$, flight list $F$
  1.  for all flight $f \in F$
        (a) if not given, assign runway and gate
        (b) find shortest legal route $r^{*f} \in R(f)$
      end for
  2.  for all flight $f \in F$
          find all non-shareable resources, and generate precedence sequences for using
          these. If there are more than one sequence to be tested at a resource, test
          one by one
      end for
  3.  Given a route for each flight $f \in F$ and precedence constraints at each non-
      shareable resource: create a routegraph $G = (\mathscr{V}, \mathscr{E})$ using the instuctions on page
      40. Note: If a routegraph already has been given, it may be updated according
      to the discussion in section 5.4. Then, skip step 4.
  4.  Find an initial dual solution using the initial dual solution algorithm (Algo-
      rithm 7, page 51).
  5.  Solve the scheduling problem using the network simplex algorithm. If the initial
      solution is primal feasible, choose the primal simplex algorithm. If dual feasible,
      choose the dual network simplex algorithm.
          • If there are more than one sequence to be tested in step 2, repeat step 3 -
            4 for each alternative, and choose the one with the best result
          • When new information appear, repeat step 1- 5.
```

5.7 Algorithm performance

In this section we will discuss the performance of Algorithm 8. In computer science, the usual way to do this is to count the number of steps it takes to complete the problem and present the result using Big-O-notation.

5.7.1 Big-O-notation

The Big-O-notation is used to give an upper bound on the asymptotic behaviour of a function. The Big-O-notation has the following definition:

Definition 5.6 (Big-O-notation)**.** If $f(n) = O(g(n))$, then there exists constants $c, N > 0$ such that $f(n) \leq c(n)$ for all $n \geq N$.

When we say that an algorithm is $O(g(n))$, we mean that the algorithm is a member of the set $O(g(n))$. By this we mean that the time it takes for the algorithm to complete the problem, in worst case, is bounded by this expression $g(n)$, where n is the size of our input (e.g. the number of edges in a graph).

5.7.2 Performance of the main algorithm

The main algorithm calls on several other algorithms, so the total performance will depend on how each step are being solved. In this section we will refer both to the Arlanda airport graph ($G = (V, E)$) and the routegraph ($\mathscr{G} = (\mathscr{V}, \mathscr{E})$).

Since gate and runway often has been allocated in the flight schedule, the performance of the first three steps is the sum of finding the shortest path for each flight, finding all non-shareable resources and constructing the routegraph. The number operations in the two last steps will depend on the total route length we find in the first. For finding all non-shareable resources, we need to go through the list of resources each flight will use. The length of this list is less than $|\mathscr{V}|$, since $\mathscr{V}$ also includes delay and early vertices.

As noted earlier in this chapter, any shortest path algorithm can be used for finding the shortest legal route for each flight $f \in F$. In chapter 2 we described two different algorithms for solving this, Dijkstra and Bellman-Ford algorithm, where Dijkstra is the fastest one. In fact, we can achieve a running time of $O(|V| \log(|V|) + |E|))$ [5, page 599].

The performance of the Bellman-Ford algorithm depends on the how many times the set of edges is visited. Its worst case performance is given by $O(|V||E|)$. If all vertices are reachable from the root, this will also be the worst case performance of the initial dual solution algorithm described earlier in this chapter, since the algorithm will return a tree solution after the first two steps.

If not all vertices are reachable from the root vertex, Algorithm 7 will perform the Bellman-Ford algorithm on multiple disjoint trees. Since the trees are disjoint, the complexity for these steps is still $O(|V||E|)$. However, step 4 also involves operations for combining the disjoint trees. $\delta = \min(c_{u,v} - y_v + y_u)$ takes 2 additions for all edges in the cut between V_1 and V_2, and $y_i = y_i' - \delta$ requires 1 addition for all vertices in V_2. We do not know in advance how many subtrees, so the worst case performance for this will be $O(|E|)$ for computing the residual cost, and $O(|V - 1|)$ for updating the tree. In total, a worst case performance of algorithm 7 is $O(|V||E|) + O(|V - 1|) + O(|E|)$, where $O(|V||E|)$ will come to dominate.

For the network simplex algorithm, it is more difficult to give a good approximation on the performance. Due to degeneracy, the algorithm may cycle, and even when it does not, there is no polynomial bound on the computations. We can however give a rough count of the computational requirements for each iteration:

- It take $O(|\mathscr{V}|)$ computations to evaluate the dual vector $(\overline{Y})$
- It take $O(|\mathscr{E}|)$ computations to evaluate all the reduced costs $(\overline{Z})$
- It take $O(|\mathscr{V}|)$ computations to effect the change of basis

However, the running time for the network simplex algorithm can be improved by using more effective ways to update the variables or using scaling algorithms.

Finally, the most time-consuming in our algorithm may be the number of sequences we would like to explore. For instance, for 10 flights using the same resource (e.g. the runway) there are $10! = 3628800$ possible sequences. Therefore, when the number of flights increases, we need good heuristics to find suitable sequences, and not all possible ones.

Chapter 6

Test Runs

Throughout this thesis we have worked with a model for representing the scheduling problem at an airport. In the previous chapter we described an algorithm that can compute a feasible schedule. In this chapter we describe some experiments and test runs using this algorithm.

5 main sets of experiments are described in this chapter. Together they illustrate the behaviour of the problem when different weights are given to the two objectives. The effect of changing the weights are important as an analysis of this could aid in understanding how the two objectives work together. This in turn may improve the realized schedule and reduce the workload for the controllers.

We will start this chapter by presenting the actual configuration used in , before we analyse how the schedule is affected by adjusting the weights. In section 6.2.2 we will focus on the time changes in our sequence when we revise, and then we look at the correspondence between the objective function and sequence changes in 6.2.3. We will end this chapter with a small summary of the results in section 6.3.

6.1 Experimental configuration

The aim in this section is to detail the actual configuration used for the experiments.

Dataset

The input data for the experiments can be found in Appendix B, while some details are presented in table 6.1.

Time unit

We will use seconds as time unit, and we will use the start time of the time horizon H as a reference point. For dataset 1 and 2 the time horizon starts at 07:00:00, meaning that

Dataset	Runway	Nr. of Aircraft	Nr. of Arrivals
1	19R	10	6
2	19R	10	7

Table 6.1: Dataset details

7 am is the reference point for these datasets. So, when a flight f has a target time at 07:15:00, we say that the target time for flight f is 900 (15*60).

Simulation step size

A step size of one minute was used in these experiments. Since the opportunities for resequencing are limited when there are flights in the manoeuvring area, we have assumed that all flights are still airborne or parked at the gate when the resequencing took place.

Milestones

As milestones we will use landings for arrivals and take-offs for departures, with the corresponding target landing time and target take-off time.

Target window

Due to small datasets, we have chosen to use a smaller target window than the controllers usually work with. The target window will start 2 minutes before the target time and end 3 minutes after ($[T_i^f - 120, T_i^f + 180]$), where T_i^f is the target time for flight f at milestone i. The target window bounds are given by $l_i = 120, u_i = 180$.

Sequence stability

Sequence stability is not included in the linear programming model presented in Chapter 4. In this chapter we will first present results where sequence stability is not considered, before this is included in later experiments.

Separation rules

The separation rules on the runway were as in table 4.2.

6.2 The effect of changing the weights

There are 6 weights, for each flight f in the objective function which can be changed in the model and in simulation Here, we will first repeat some of the notation presented in

Chapter 4, before we present experimental results.

- $b_i^{\epsilon f}$ - The cost per time unit of flight f being early at resource i
- $b_i^{\delta f}$ - The cost per time unit of flight f being late at resource i
- $b_{li}^{\epsilon f}$ - The cost per time unit of flight f being earlier than $T_i^f - l_i^f$ at resource i
- $b_{ui}^{\delta f}$ - The cost per time unit of flight f being later than $T_i^f + u_i^f$ at resource i
- $b_i^{\varepsilon f}$ - The cost per time unit of changing the time flight f enter resource i
- $b_i^{\zeta f}$ - The cost per time unit of changing the time flight f enter resource i

When presenting the results, we will refer to a cost vector of the weights in the objective function, B^f, where $B^f = (b_i^{\epsilon f}, b_i^{\delta f}, b_{li}^{\epsilon f}, b_{ui}^{\delta f}, b_i^{\varepsilon f}, b_i^{\zeta f})$. Since we do not distinguish between flights, we will use the same cost vector for all flights.

When presenting the results, we will use the following notation:

- $\sum_{f \in F} \sum_{i \in P_M^f} |\delta_i^f + \epsilon_i^f|$ - Total deviation from target times (in seconds)
- $\sum_{f \in F} \sum_{i \in P_M^f} |\delta_{ui}^f + \epsilon_{li}^f|$ - Total deviation outside the target window (in seconds)
- $\sum_{f \in F} \sum_{i \in P_M^f} |\zeta_i^f + \varepsilon_i^f|$ - Total deviation from the previous solution (in seconds)
- Total Cost - The total cost
- Cost P - Cost associated with not being punctual
- Cost S - Cost associated with not being stable

Experiments

In section 4.4.3 we described several variants of the cost function, and the experiments in this chapter are based on these. The first three experiments are carried out without considering sequence stability. In experiment 1 - 4, one rescheduling step is carried out, while the schedule is revised three times in experiment 5.

In first two experiments, deviation from target times are penalised with a cost term as given in figure 4.6c, meaning that there is an interval around the target time which will not give any cost. For stability we will use a cost term with same form as in figure 4.6a. For both objectives, a deviation will give the same cost regardless of the deviation is positive or negative, that is $b_i^{\delta f} = -b_i^{\epsilon f}, b_i^{\zeta f} = -b_i^{\varepsilon f}$.

In the third experiment deviation from target times are penalised with a cost term as given in figure 4.6b. The stability cost term are still as in figure 4.6a.

In the fourth experiment, sequence stability will be considered. We will use the same cost terms as in experiment one and three, with only few exceptions.

The final experiment does not have a general setup. Here we will use a collection of different cost vectors and revise the schedule three times.

6.2.1 Experimental results

In our first experiment we used dataset 2 as an input and we only penalized deviation outside the target window ($B = (0, 0, 0, 0, -100, 100)$), see figure 4.6c.

	Type	Target Time for Arrival	Planned Arrival Time	Target Take-Off Time	Planned Take-Off Time
Flight 1	D	-	-	1200	1374
Flight 2	A	1320	1390	-	-
Flight 3	A	1440	1470	-	-
Flight 4	A	1500	1550	-	-
Flight 5	D	-	-	1500	1654
Flight 6	A	1680	1670	-	-
Flight 7	A	1740	1750	-	-
Flight 8	A	1800	1830	-	-
Flight 9	D	-	-	1800	1934
Flight 10	A	1860	1950	-	-

Table 6.2: Optimal Initial solution - Dataset 2 with cost vector $B = (0, 0, 0, 0, -100, 100)$

	Type	Target Time for Arrival	Planned Arrival Time	Target Take-Off Time	Planned Take-Off Time
Flight 1	D	-	-	1200	1380
Flight 2	A	1320	1276	-	-
Flight 3	A	1440	1447	-	-
Flight 4	A	1500	1567	-	-
Flight 5	D	-	-	1500	1551
Flight 6	A	1680	1664	-	-
Flight 7	A	1740	1904	-	-
Flight 8	A	1800	1744	-	-
Flight 9	D	-	-	1800	2008
Flight 10	A	1860	1824	-	-

Table 6.3: Optimal Recomputed solution - Dataset 2 with cost vector $B = (0, 0, 0, -100, 100)$

The initial optimal solution is given in table 6.2 while the optimal recomputed solution is given in table 6.3. If we look at these two schedules, almost 12 minutes (709 seconds) deviates the planned landing and take-off times in the two solutions. In addition, 9 flights got a new leading flight.

Experiment 1

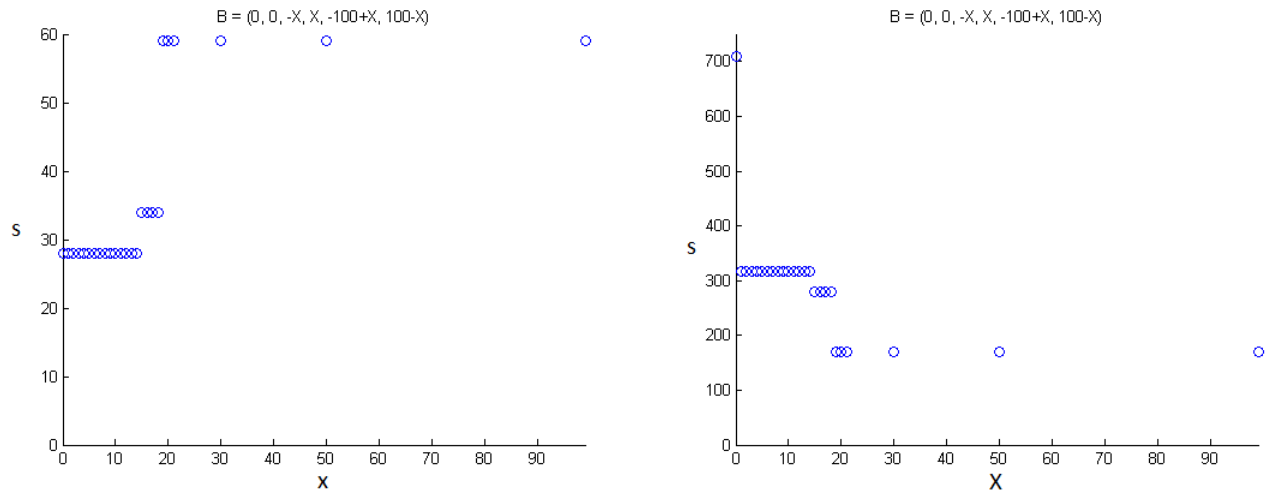
In our first attempt to avoid deviations as presented above, we first added a cost of deviation from the previous value (introduced time stability cost term). Since this term only affect the rescheduling part, we got the same initial solution as in table 6.2 for all X. We let $B = (0, 0, -X, X, -100+X, 100-X)$, meaning that the cost term for punctuality decreases when the stability term increase. We let X varies between 0 and 99. When $X = 0$, we will get the solution given above. The results are given in table 6.4 and illustrated in figure 6.1.

X	$\sum_{f\in F}\sum_{i\in P_M^f} \lvert\delta_i^f + \epsilon_i^f\rvert$	$\sum_{f\in F}\sum_{i\in P_M^f} \lvert\delta_{ui}^f + \epsilon_{li}^f\rvert$	$\sum_{f\in F}\sum_{i\in P_S^f} \lvert\zeta_i^f + \varepsilon_i^f\rvert$	Total cost	cost S	Cost P
0	829	28	709	2800	0	2800
1	856	28	316	3088	316	2772
5	856	28	316	4240	1580	2660
8	856	28	316	5104	2528	2576
10	856	28	316	5680	3160	2520
11	856	28	316	5968	3476	2492
12	856	28	316	6256	3792	2464
13	856	28	316	6544	4108	2436
14	856	28	316	6832	4424	2408
15	892	34	280	7090	4200	2890
16	892	34	280	7336	4480	2856
17	892	34	280	7582	4760	2822
18	892	34	280	7828	5040	2788
19	892	59	170	8009	3230	4779
20	892	59	170	8120	3400	4720
21	892	59	170	8231	3570	4661
30	892	59	170	9230	5100	4130
40	892	59	170	10340	6800	3540
50	892	59	170	11450	8500	2950
99	892	59	170	16889	16830	59

Table 6.4: Solution data after one resequencing - Dataset 2 with cost vector $B = (0, 0, X, -100+X, 100-X)$

In figure 6.1a we have plotted how much the reoptimised schedule deviates from the target windows, while we have plotted how much this solution deviates from the initial one in figure 6.1b. We let X increase along the x-axis, while the total deviation in seconds is given along the y-axis. From figure 6.1 it seems like there is a trade-off between punctuality and stability. When we increase X, the new solutions deviate less from the previous one, while the punctuality decreases and the solutions deviate more from the target window.

Observe in table 6.4, that by introducing a cost for deviating from the previous solution in the objective function ($X > 0$), we get a solution that is far more stable than when this is not considered ($X = 0$). The deviation between the schedules decreases from 709 seconds to 316, meaning that the deviation is reduced with $55,4\%$.



(a) Punctuality - deviation from target window in sec (b) Stability - deviation from initial solution in sec

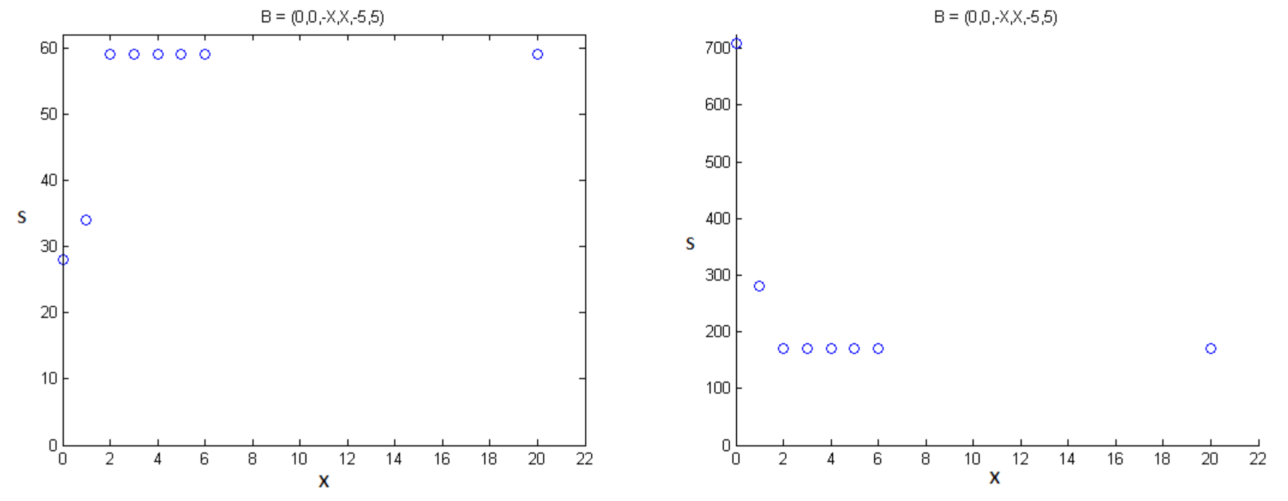
Figure 6.1: Coherence between the X-value and deviations

Experiment 2

In our next approach, we let the cost term for punctuality be constant, while we are using different values for the stability term. This gives us the general cost vector $B = (0, 0, -X, X, -5, 5)$, for some $X \in [0, 20]$. The results from this experiment are given in table 6.5 and in figure 6.2.

X	$\sum_{f\in F}\sum_{i\in P_M^f} \lvert\delta_i^f + \epsilon_i^f\rvert$	$\sum_{f\in F}\sum_{i\in P_M^f} \lvert\delta_{ui}^f + \epsilon_{li}^f\rvert$	$\sum_{f\in F}\sum_{i\in P_S^f} \lvert\zeta_i^f + \varepsilon_i^f\rvert$	Total cost	Cost S	Cost P
0	829	28	709	140	0	140
1	892	34	280	450	280	170
2	892	59	170	635	340	295
3	892	59	170	805	510	295
4	892	59	170	975	680	295
5	892	59	170	1145	850	295
6	892	59	170	1315	1020	295
20	892	59	170	3695	3400	295

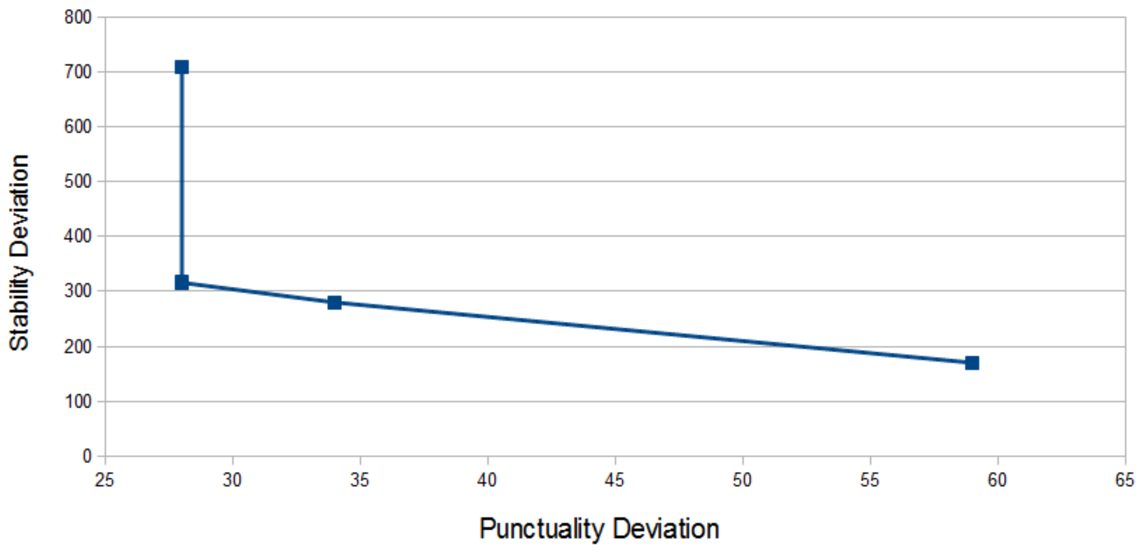
Table 6.5: Solution data after one resequencing - Dataset 2 with cost vector $B = (0, 0, X, -5, 5)$



(a) Punctuality - deviation from target window in sec (b) Stability - deviation from initial solution in sec

Figure 6.2: Coherence between the X-value and deviations

Also here we can observe the same trade-off tendency as in the previous example, and there are some results appearing in both experiments above, plotting these will give us the following curve:



-

Figure 6.3: Pareto front, experiment 1 and 2

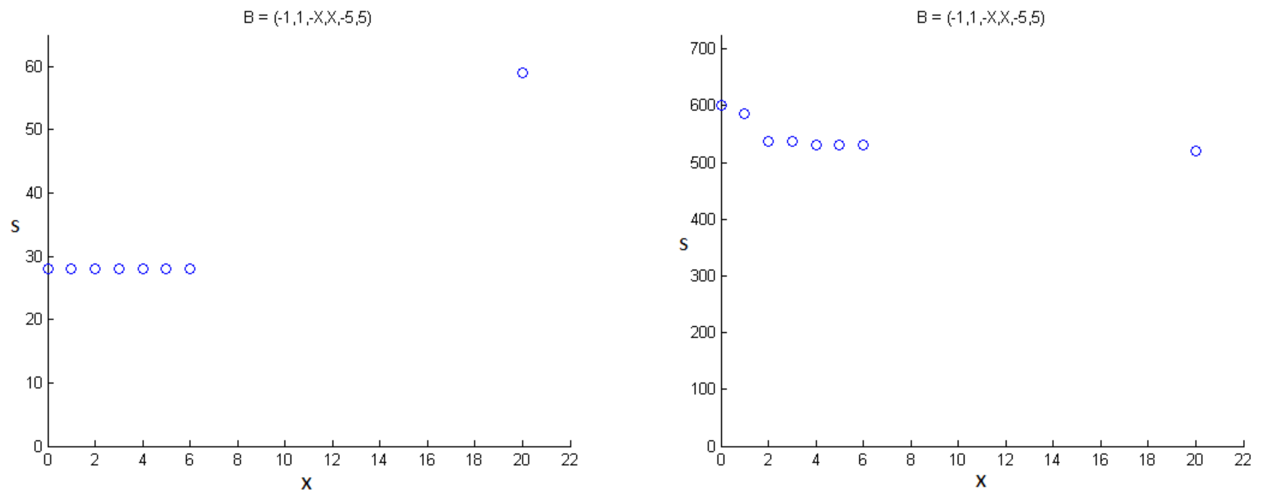
In figure 6.3 we see that you can not improve stability without depress the punctuality. However, the real effect of changing the weights is difficult to measure since the algorithm using these cost vectors, does not favor between solution values that lies inside the target window. This is investigated in the next experiment.

Experiment 3

Next we add a value to $b_i^{\delta f}$ and $b_i^{\epsilon f}$, meaning that we will penalize any deviation from the target times, not only outside the target window. This will give us a cost term of not being punctual as in figure 4.6b. For this experiment we are using the cost vector $B = (-1, 1, -X, X, -5, 5)$, for some $X \in [0, 20]$. The rescheduling solutions from this experiment are given in table 6.6 and illustrated in figure 6.5.

X	$\sum_{f\in F}\sum_{i\in P_M^f} \lvert\delta_i^f + \epsilon_i^f\rvert$	$\sum_{f\in F}\sum_{i\in P_M^f} \lvert\delta_{ui}^f + \epsilon_{li}^f\rvert$	$\sum_{f\in F}\sum_{i\in P_S^f} \lvert\zeta_i^f + \varepsilon_i^f\rvert$	Total cost	Cost S	Cost P
0	530	28	602	670	0	670
1	530	28	586	1256	586	670
2	579	28	537	1793	1074	719
3	579	28	537	2330	1611	719
4	596	28	532	2864	2128	736
5	596	28	532	3396	2660	736
6	596	28	532	3928	3192	736
20	616	59	520	11311	10400	911

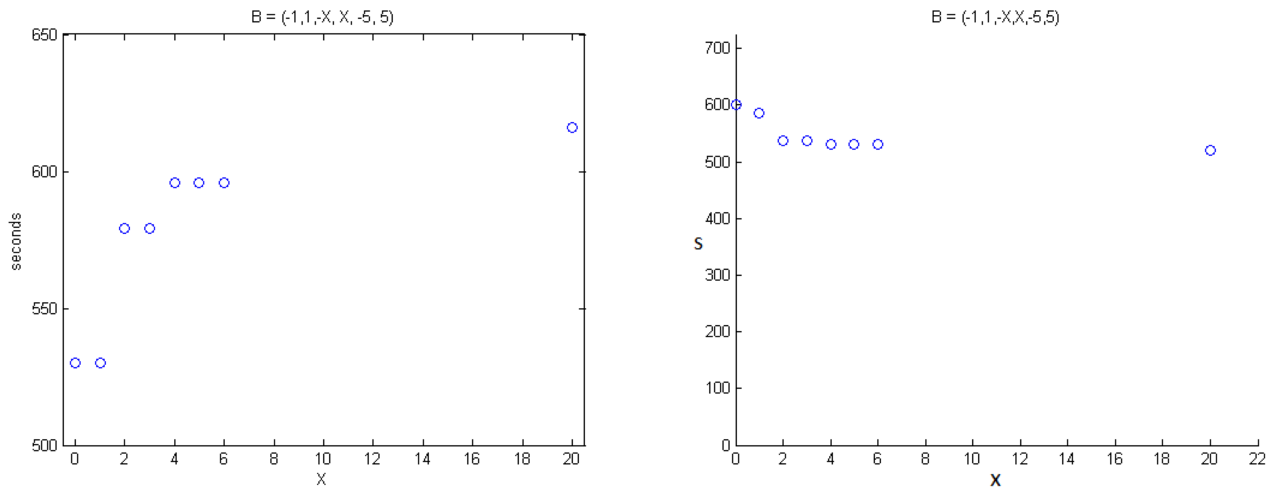
Table 6.6: Solution data after one resequencing - Dataset 2 with cost vector $B = (-1, 1, -X, X, -5, 5)$



(a) Punctuality - deviation from target window in sec (b) Stability - deviation from initial solution in sec

Figure 6.4: Coherence between the X-value and deviations

In this example, it could be better to compare stability against the total deviation from target times. The reason is that in this example, any deviation is penalised, while we in the two previous examples only penalised deviation outside the target windows.



(a) Punctuality - devition from target in sec (b) Stability - deviation from inital solution in sec

Figure 6.5: Coherence between the X-value and deviations

Also in this example it seems to be a trade-off between stability and punctuality, but the changes are far less than in the two previous examples. The trade-off can also be seen in figure **??**, where we have plotted the deviation from the previous solution against the total deviation from target times.

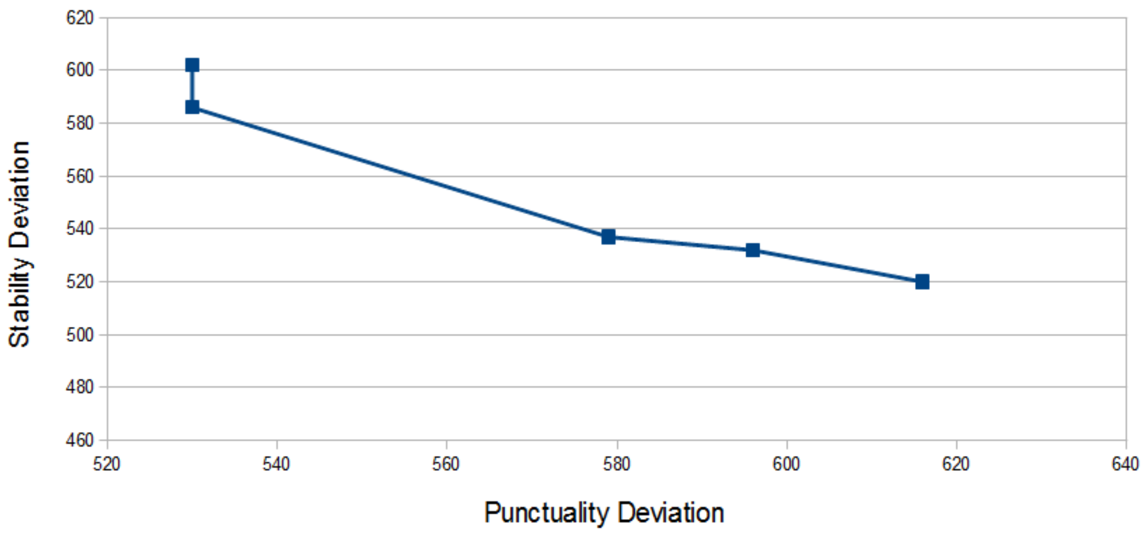


Figure 6.6: Pareto front

Experiment 3 where also performed using dataset 1 as input. It gave the following result where we also can se the trade-off tendency:

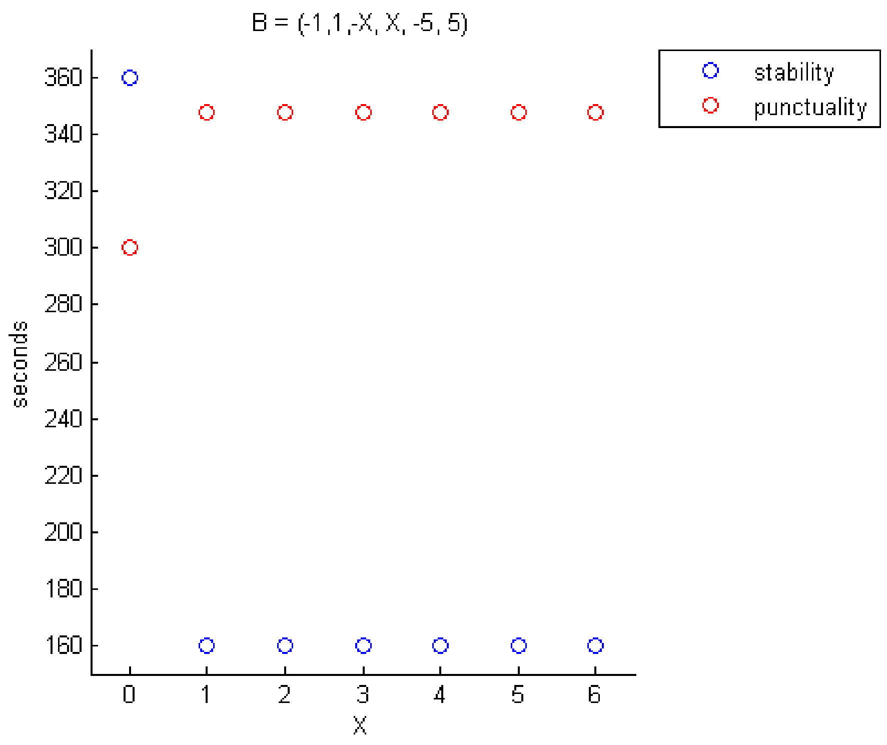


Figure 6.7: Coherence between the X-value deviations, using Dataset 1 and cost vector $B = (-1, 1, -X, X, -5, 5)$

Experiment 4

Here we have repeated experiment 1, but also considered sequence stability. The results given in table 6.7 are almost equal to the results in table 6.4. The most important difference occur when $X = 0$. In the experiment with sequence stability, we got a solution with 32% less deviation in seconds from the previous solution than in experiment 1.

X	$\sum_{f\in F}\sum_{i\in P_M^f} \lvert\delta_i^f + \epsilon_i^f\rvert$	$\sum_{f\in F}\sum_{i\in P_M^f} \lvert\delta_{ui}^f + \epsilon_{li}^f\rvert$	$\sum_{f\in F}\sum_{i\in P_S^f} \lvert\zeta_i^f + \varepsilon_i^f\rvert$	Total cost	Cost S	Cost P
0	752	28	476	2804	0	2800
1	856	28	316	3092	316	2772
5	856	28	316	4244	1580	2660
10	856	28	316	5684	3160	2520
20	892	59	170	8122	3400	4720
21	892	59	170	8233	3570	4661
30	892	59	170	9232	5100	4130
50	892	59	170	11452	8500	2950
99	892	59	170	16891	16830	59

Table 6.7: Solution data after one resequencing - Dataset 2 with cost vector $B = (0, 0, -X, X, -100 + X, 100 - X)$ and sequence stability

Here we have repeated experiment 3, but also considered sequence stability. The results given in table 6.8. Also here, the results are almost similar to those in table 6.6.

X	$\sum_{f\in F}\sum_{i\in P_M^f}\lvert\delta_i^f+\epsilon_i^f\rvert$	$\sum_{f\in F}\sum_{i\in P_M^f}\lvert\delta_{ui}^f+\epsilon_{li}^f\rvert$	$\sum_{f\in F}\sum_{i\in P_S^f}\lvert\zeta_i^f+\varepsilon_i^f\rvert$	Total cost	Cost S	Cost P
0	530	28	602	674	0	670
1	530	28	586	1260	586	670
2	579	28	537	1797	1074	719
4	579	28	537	2871	2148	719
5	596	28	532	3403	2660	736
6	596	28	532	3935	3192	736

Table 6.8: Solution data after one resequencing - Dataset 2 with cost vector $B = (-1, 1, -X, X, -5, 5)$ and sequence stability

Experiment 5 - Changes in the sequence

In the experiments above, we have seen that there is a trade-off tendency between stability and punctuality. In addition to this, it is also interesting to compare how the sequence changes when there is more than one resequencing. In figure 6.8 - figure 6.11 we have illustrated how the sequence evolve for 4 different setups.

The runway sequencing is worst when not including any stability distance function and only penalising deviation outside the target window (figure 6.8). In the three other setups, the sequence is much more stable. There are however difficult to say whether one of these three approaches is better than the other, since the total sequence changes is the same.

6.3 Result summary

In all the experiments presented in this chapter, it seems to be a trade-off between stability and punctuality. When the cost of deviating from the previous value increases, the time stability increases, while the punctuality decreases. The results also indicates that including stability in the objective function, greatly improves the stability without a major decrease in punctuality, illustrated by the steep drop in the pareto curves (figure 6.3 and figure 6.6). However, this effect should be investigated further to give more precise conclusions.

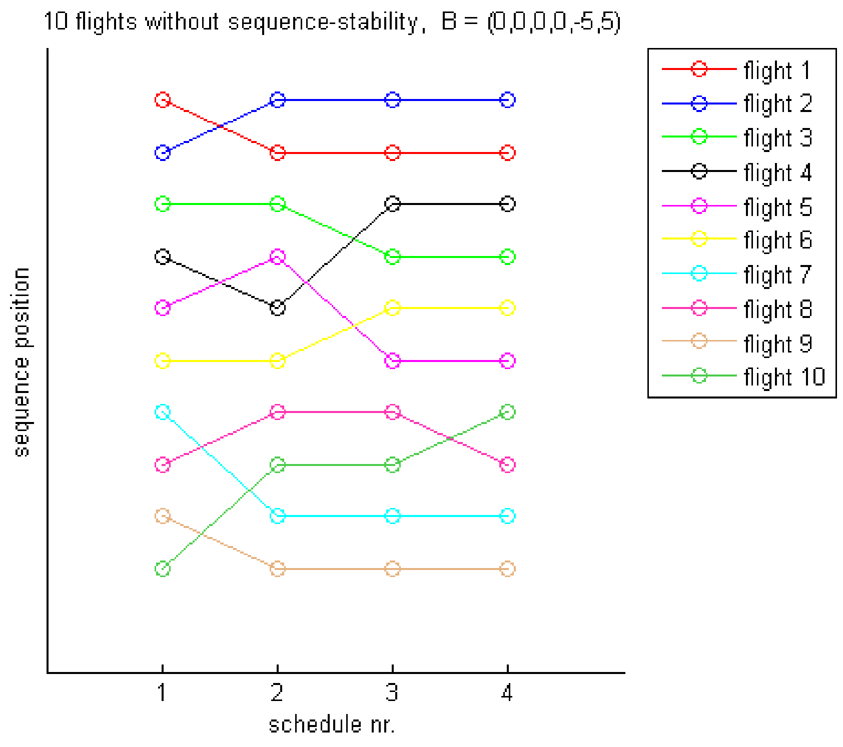


Figure 6.8: Picture of runway sequencing, $B = (0, 0, 0, 0, -5, 5)$

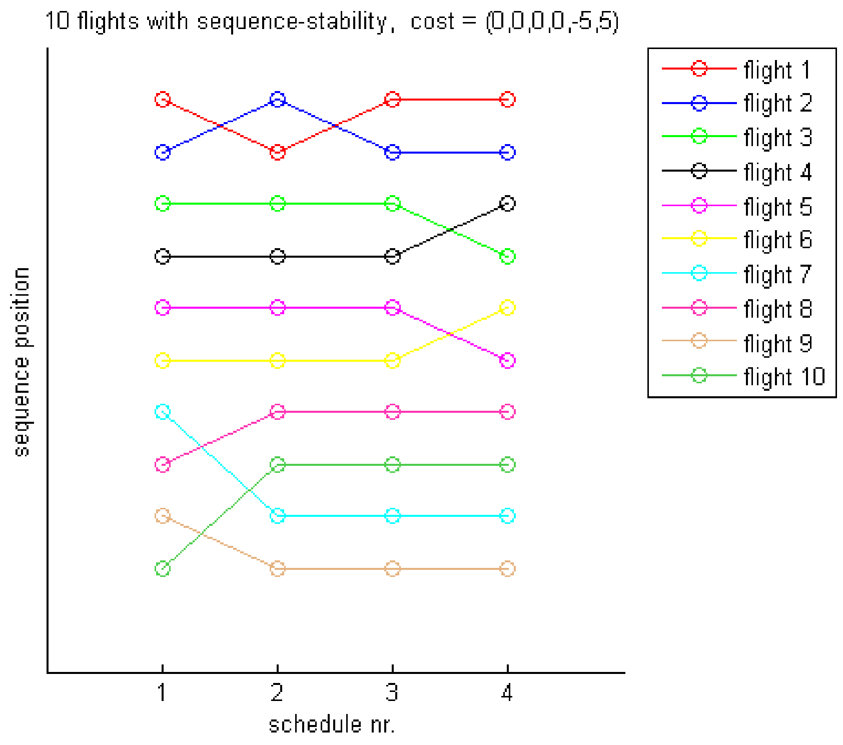


Figure 6.9: Picture of runway sequencing, $B = (0, 0, 0, 0, -5, 5)$, with sequence stability

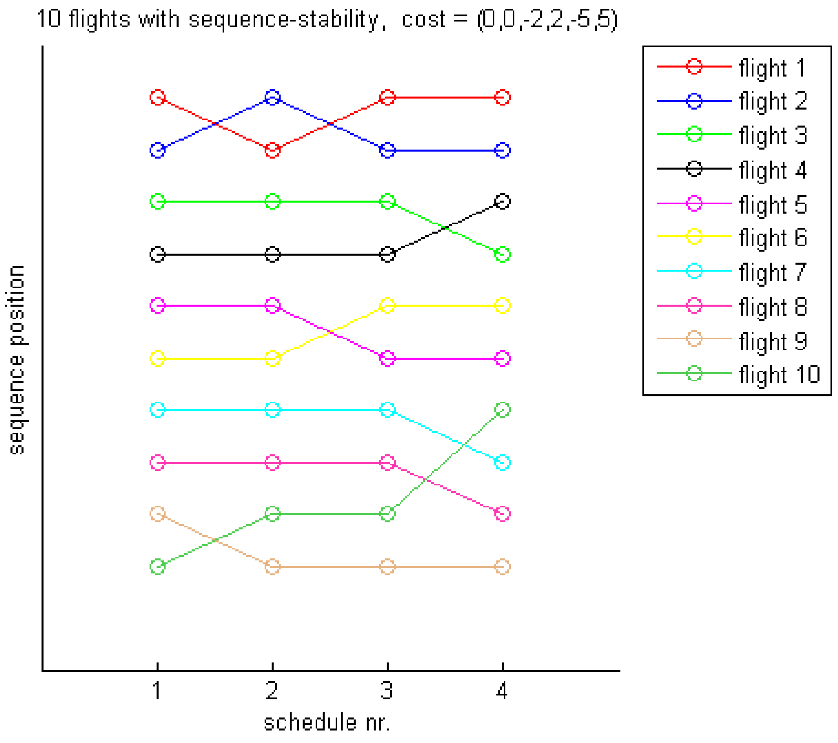


Figure 6.10: Picture of runway sequencing, $B = (0, 0, -2, 2, -5, 5)$, with sequence stability

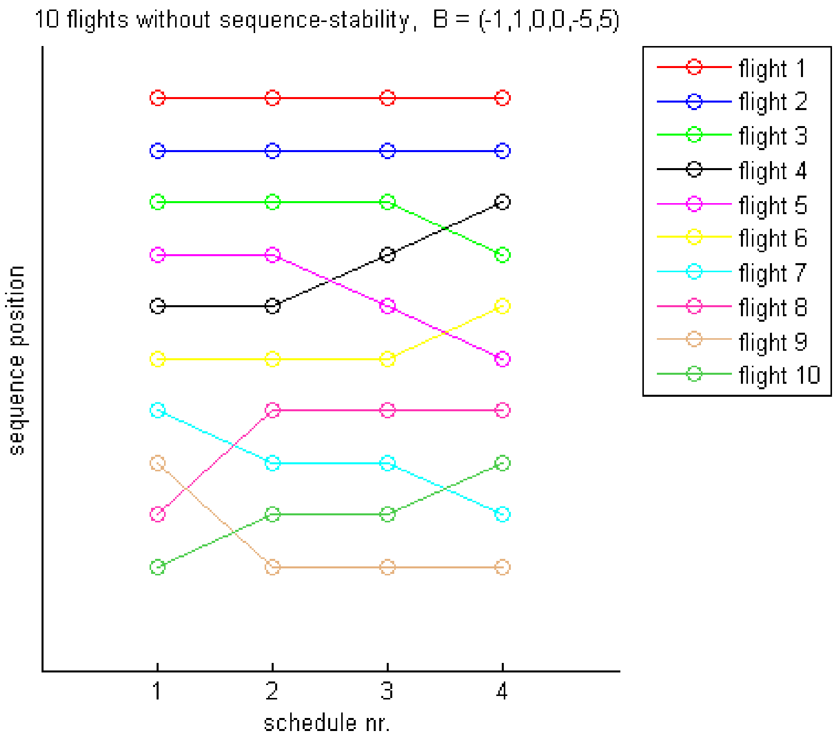


Figure 6.11: Picture of runway sequencing, $B = (-1, 1, 0, 0, -5, 5)$

Chapter 7

Summary and further research

In the introduction we introduced ATM operations and their need for stable solutions. We also presented the goal of the project. After covering some background theory in chapter 2, we presented a small survey on literature on scheduling and stability in chapter 3. In chapter 4 we established an LP formulation of the scheduling problem under a given context, where the flight routes and precedence constraints were known. In chapter 5 we implemented this linear program using a minimum cost flow solver. We also presented the main algorithm. Finally, in chapter 6 we performed simulations and presented experimental results.

With no earlier experience using *C#*, a great amount of time have been spend trying to learn the language. The first algorithms were implemented using matrix notation, and worked fine for 1-6 flights, which was the first thing we tested. However, when the we looked at larger instances with more flights, the computational performance was not good enough. Due to lack of time, there was no time to re-implement these using more efficient data structures.

From the results in chapter 6, it does seem like there is a trade-off between punctuality and stability. When the cost of deviating from the previous value increases, the time stability increases while the punctuality decreases. This can be seen in all our experiments. Especially in experiment 1 and 2, the solutions are far more time stable as fast as we include stability in the objective function. This trend can also be seen in the other experiments, but the effect is smaller.

However, we have only investigated 2 datasets and performed 5 types of experiments, so we should be careful to draw any conclusion before we have investigated this problem further.

7.1 Further research

In the experiments performed in chapter 6, a lot of time goes to generating all possible sequences and test whether they are feasible or not. Finding a good heuristic that generate feasible sequences at non-shareable resources, instead of enumerating all theoretically possible ones, would improve the computational workload and we would be able to test greater and more complicated datasets.

In addition to use larger dataset, it would be interesting to do some experiments where tardiness and earliness are penalised differently, using different weights in piecewise linear terms. This might give us more understanding about how the different weights affect the recomputations.

An interesting extension to the model would be to include more of the in-flight part of the operations or even the whole flight, including capacities on the origin- and destination airports. Since there are specific separation rules in the air, this would give us a more accurate model. An other extension would be to not use fixed running times between two airport resources, but instead use more flexible upper and lower bounds.

With a more accurate model and more efficient implementation of the algorithm, we would be able to perform more advanced experiments. Doing this would lead us one step closer to finding good values for the weights in the objective function and drawing more precise conclusions.

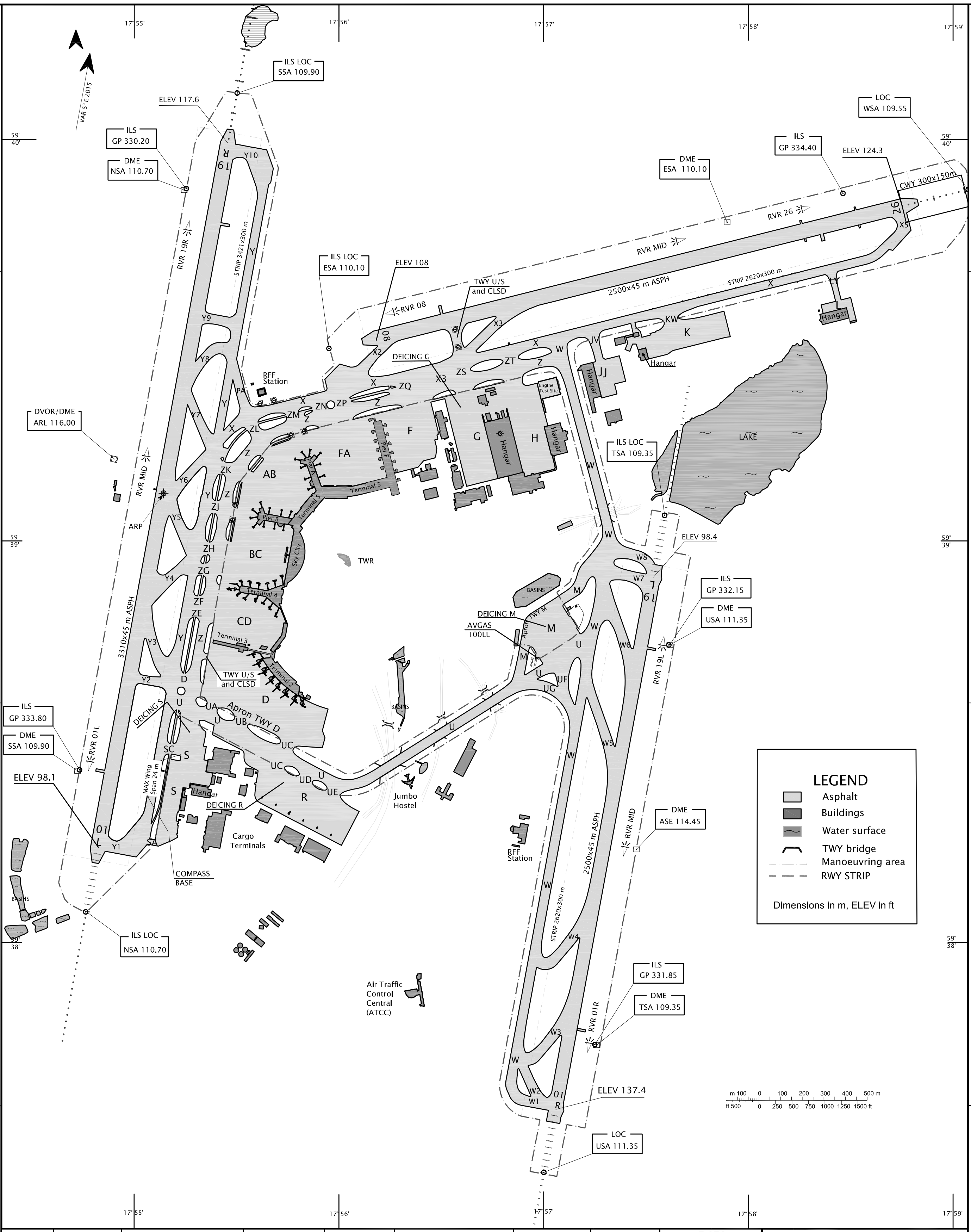
Appendix A

Arlanda Information

The airport we chose to use as a case study for this thesis is Stockholm-Arlanda. The airport is the largest one in Sweden, located about 40 km north of Stockholm. As of Mars 2014, it has 3 runways and 4 terminals and the traffic density are described as medium to heavy

The movements on the ground are mainly decided by the characteristic and restrictions on the taxiways. Some of the taxiways are one-way directed, meaning that both arriving and departing aircraft must traverse them in the same defined direction. There are some segments only for arrivals and segments used only for departures. Complementary information about the taxiway structure can be found in the following charts. As can be seen from these, the normal taxi-route procedure, both for arrival and departures, is clockwise taxiing where parallel taxiways are established [9].

Change: THR ELEV, RFF stations, DME



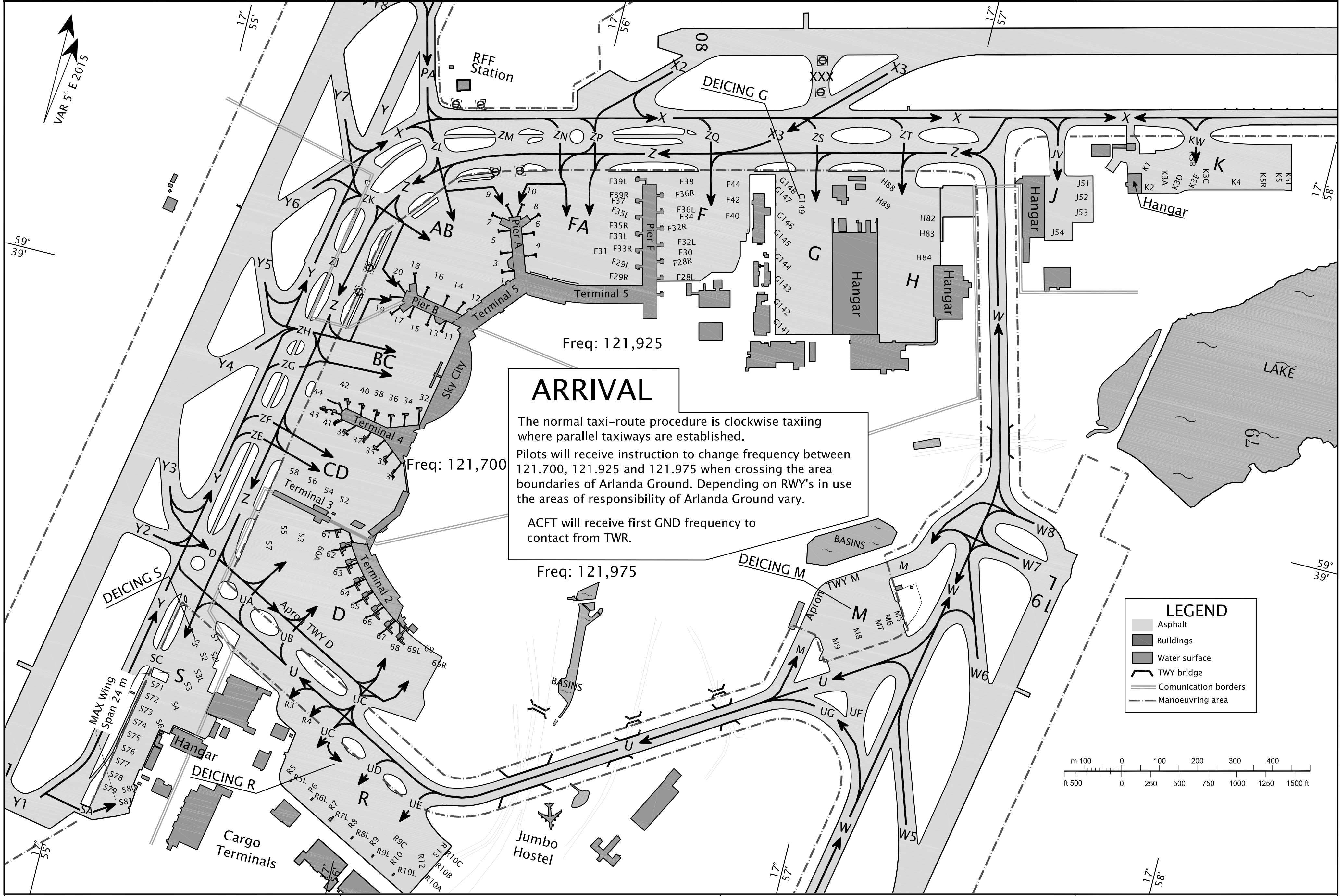
RWY	BRG MAG	THR COORDINATES	THR ELEV (FT)	TDZ ELEV (FT)	RWY DIMENSIONS (m)	STRENGTH PCN	SURFACE	PAPI GP	PAPI MEHT
01L	005°	593814.12N 0175447.61E	98.1	100	3301x45	120 F/A/X/T	ASPH	Left 3.0°	61.4 ft 18.7 m
19R	185°	593959.05N 0175525.56E	117.6	118	3301x45	120 F/A/X/T	ASPH	Left 3.0°	56.4 ft 17.2 m
01R	005°	593735.03N 0175702.67E	137.4	137	2500x45	90 F/B/X/T	ASPH	Right 3.0°	57.3 ft 17.5 m
19L	185°	593854.48N 0175731.49E	98.4	98	2500x45	90 F/B/X/T	ASPH	Left 3.0°	57.3 ft 17.5 m
08	071°	593930.31N 0175610.08E	108		2500x45	90 F/B/X/T	ASPH	Left 3.0°	56.4 ft 17.2 m
26	251°	593950.03N 0175844.95E	124.3	124	2500x45	90 F/B/X/T	ASPH	Left 3.0°	60.0 ft 18.3 m

AD GEOGRAFICAL DATA

AD ELEV 137ft
ARP 59°39'07''N 017°55'07''E
For further information see ESSA AD 2.2

REMARKS

RWY and TWY Markings see AD 2-ESSA-2-3
RWY and TWY Lightning see AD 2-ESSA-2-4
Altimeter Check-Location (ACL) and stand coordinates see AD 2-ESSA ACL/INS Reference points

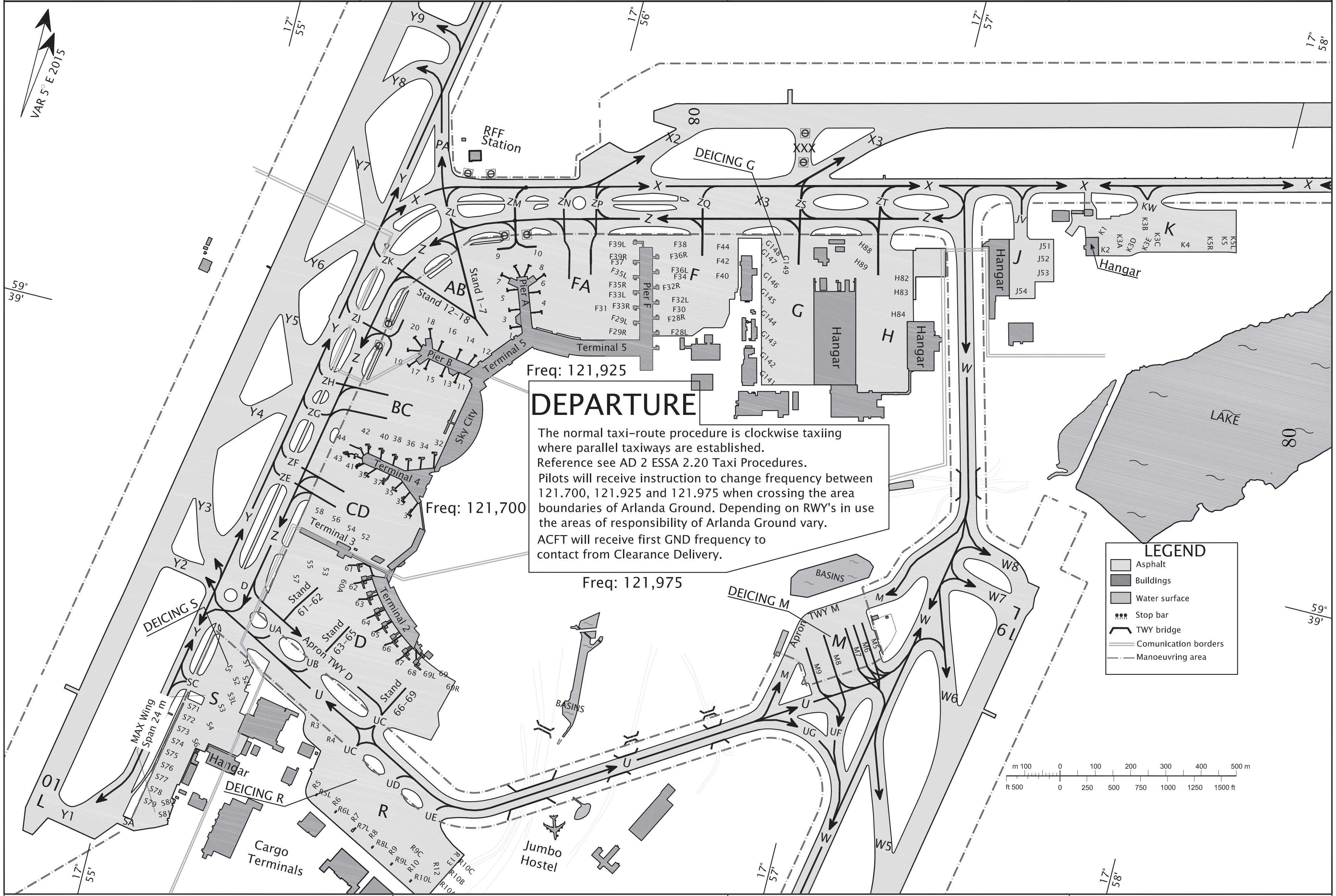


ARRIVAL

The normal taxi-route procedure is clockwise taxiing where parallel taxiways are established.

Pilots will receive instruction to change frequency between 121.700, 121.925 and 121.975 when crossing the area boundaries of Arlanda Ground. Depending on RWY's in use the areas of responsibility of Arlanda Ground vary.

ACFT will receive first GND frequency to contact from TWR.



DEPARTURE

The normal taxi-route procedure is clockwise taxiing where parallel taxiways are established.
Reference see AD 2 ESSA 2.20 Taxi Procedures.
Pilots will receive instruction to change frequency between 121.700, 121.925 and 121.975 when crossing the area boundaries of Arlanda Ground. Depending on RWY's in use the areas of responsibility of Arlanda Ground vary.
ACFT will receive first GND frequency to contact from Clearance Delivery.

Appendix B

Datasets

B.1 Dataset 1

Flight	Type	Gate	Runway	TOBT	TLT/TTOT	TIBT
1	D	7	19R	07:05:00	07:15:00	
2	D	68	19R	07:05:00	07:15:00	
3	A	10	19R		07:17:00	07:27:00
4	A	57	19R		07:18:00	07:28:00
5	A	44	19R		07:19:00	07:29:00
6	D	62	19R	07:10:00	07:20:00	
7	D	4	19R	07:10:00	07:20:00	
8	A	32	19R		07:22:00	07:32:00
9	A	5	19R		07:24:00	07:34:00
10	A	15	19R		07:25:00	07:35:00

Table B.1: Dataset 1

Deviation time	Flight	Type	Gate	Runway	UOBT
07:01:00	1	D	7	19R	07:10:00
07:01:00	7	D	4	19R	07:13:15

Table B.2: Deviations Dataset 1

UOBT is an updated off-block time.

B.2 Dataset 2

Flight	Type	Gate	Runway	TOBT	TTA/TTOT	TIBT
1	D	4	19R	07:10:00	07:20:00	
2	A	32	19R		07:22:00	07:32:00
3	A	5	19R		07:24:00	07:34:00
4	A	15	19R		07:25:00	07:35:00
5	D	12	19R	07:15:00	07:25:00	
6	A	41	19R		07:28:00	07:38:00
7	A	42	19R		07:29:00	07:39:00
8	A	54	19R		07:30:00	07:40:00
9	D	35	19R	07:20:00	07:30:00	
10	A	34	19R		07:31:00	07:41:00

Table B.3: Dataset 2

Deviation time	Flight	Type	Gate	Runway	UOBT	UTA
07:01:00	1	D	4	19R	07:13:15	
07:01:00	5	D	12	19R	07:20:00	
07:01:00	7	A	42	19R		07:31:05
07:01:00	9	D	35	19R	07:26:00	
07:02:00	6	A	41	19R		07:24:55
07:03:00	3	A	5	19R		07:25:35
07:03:00	10	A	34	19R		07:29:00

Table B.4: Deviations Dataset 2

UOBT is an updated off-block time and UTA is an updated time for arrival.

Bibliography

[1] Ravindra K. Ahuja, Thomas L.Magnanti, and James B.Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, 1993.

[2] Jason Adam David Atkin. *On-line decision support for take-off runway scheduling at London Heathrow airport.* Phd. thesis, The University of Nottingham, 2008.

[3] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization.* Athena Scientific Belmont, MA, 1997.

[4] Alistair R Clark and Hannah Walker. Nurse rescheduling with shift preferences and minimal disruption. *Journal of Applied Operational Research*, 3(3):148–162, 2011.

[5] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein, et al. *Introduction to algorithms*, volume 2. MIT press Cambridge, 2001.

[6] Maria Fox, Alfonso Gerevini, Derek Long, and Ivan Serina. Plan stability: Replanning versus plan repair. In *In Proc. ICAPS*, pages 212–221. AAAI Press, 2006.

[7] Dag Kjenstad, Carlo Mannino, Tomas Eric Nordlander, Patrick Schittekat, and Morten Smedsrud. Integrated surface and departure management at airports by optimization. pages 1–5, April 2013.

[8] Dag Kjenstad, Carlo Mannino, Tomas Eric Nordlander, Patrick Schittekat, and Morten Smedsrud. Optimizing aman-sman-dman at hamburg and arlanda airport. 2013.

[9] LFV. Information and charts - arlanda. `http://www.lfv.se/sv/FPC/IAIP/AD/AD2T/ESSA/`.

[10] L.R.Ford and D.R.Fulkerson. *Flows in Networks.* Princeton Univeristy Press, 1962.

[11] Adrian Petcu and Boi Faltings. Optimal solution stability in continuous time optimization. In *IJCAI05-Distributed Constraint Reasoning workshop, DCR05.* Citeseer, 2005.

[12] TITAN. Analysis of the current situation. `http://www.titan-project.eu/library/titan/TITAN_WP1_SLO_DEL_01_v1.0_Analysis_of_the_current_situation.pdf`.

[13] SESAR Joint Undertaking. Discover sesar - why sesar. `http://www.sesarju.eu/discover-sesar/why-sesar`.

[14] Robert J. Vanderbei. *Linear Programming: Foundations and Extensions.* Springer, 2008.

[15] S David Wu, Robert H Storer, and Chang Pei-Chann. One-machine rescheduling heuristics with efficiency and stability as criteria. *Computers & Operations Research*, 20(1):1–14, 1993.