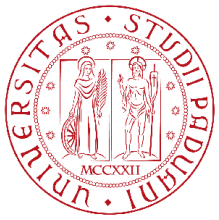


Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea Magistrale in Matematica

A mathematical programming model for air traffic flow management with horizontal and vertical separation constraints

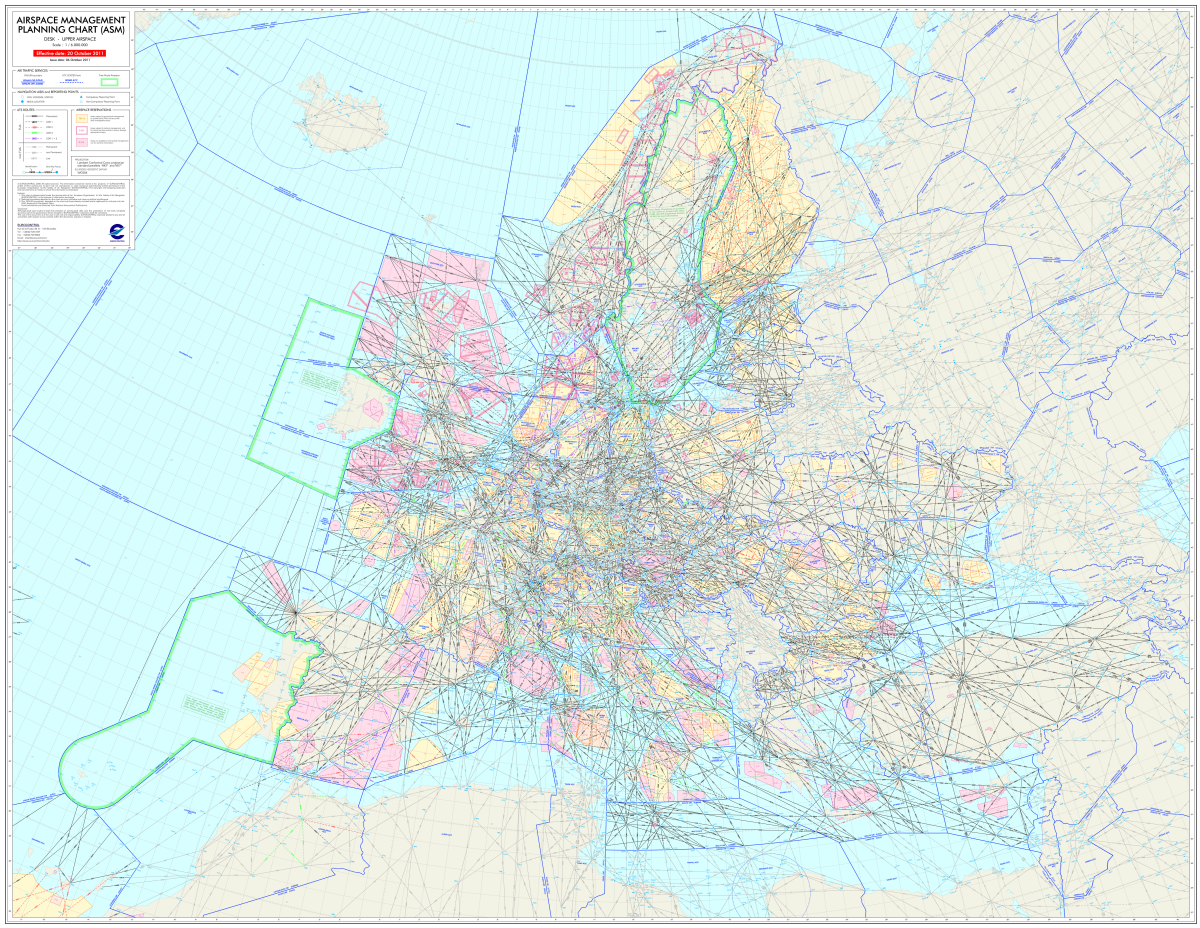
Supervisor
Prof. Luigi De Giovanni
Università degli Studi di Padova

Master Candidate
Antonio Bellon
Matriculation Nr 1132179

12 October 2018

Abstract

This thesis presents a new integer linear programming (ILP) model for the air traffic flow management (ATFM) problem. The model objective is to minimize the cost of flight delays. This issue is of primary concern in the air traffic system. The introduction provides some insight on ATFM. The problem is then addressed through a combination of flow management actions, including ground-holding, airborne-holding and distribution by levels of air traffic. Exploiting the fact that actual airspace sectors observed capacities are often higher than the nominal capacities, two types of flights separation constraints are proposed in order to increase the use of airspace without compromising safety requirements. Furthermore, in order to avoid complex trajectories, an additional class of constraints is discussed. Two classes of valid inequalities are also presented with the purpose of strengthening the underlying relaxation. The thesis subsequently reports computational experiments on small instances. The model with separation constraints is then compared to the model where no separation constraints are imposed and nominal capacities restriction are maintained. These comparisons show that the model can improve the solution for small instances. Improvements are correlated to low nominal capacities and high maximum flight level variation. Finally, implementation details are provided.



The EUROCONTROL airspace management planning chart [22]

Contents

List of acronyms

ANSP ..... Air Navigation Service Providers

ATAG ..... Air Transport Action Gruop

ATC ..... Air Traffic Controllers

ATFCM ..... Air Traffic Flow and Capacity Management

ATFM ..... Air Traffic Flow Management

ATM ..... Air Traffic Management

ATS ..... Air Traffic Services

DDR ..... Demand Data Repository

FIR ..... Flight Information Region

IATA ..... International Air Transport Association

ICAO ..... International Civil Aviation Organization

ILP ..... Integer Linear Programming

LP ..... Linear Programming

MILP ..... Mixed Integer Linear Programming

NMOC ..... Network Manager Operations Center

SM ..... Simplex Method

TBO ..... Traffic Based Operations

Chapter 1

Introduction

In this introductory chapter we expose the main motivations of our work, providing the reader with some general background on the present state of the European air traffic. Afterwards, a summary of the thesis content, highlighting major contributions, is given.

1.1 Motivations

At the present state, the European air transportation industry is without doubt one of the biggest drivers of economic growth in Europe. It has been estimated that in 2014 the aviation sector has supported more than 6.9 million jobs, of which 2.5 directly provided, and it has contributed to €531 billion in Gross Domestic Product across the European Union (EU28) [3]. During 2017, more than 890 million passengers travelled by air in Europe with an increase of 12% compared to 2008. In Italy, the number of passengers in 2006 was less than 100 million and it raised in 2016 by more than 40% [10]. On average, compared to 2016, the number of controlled flights in the EUROCONTROL area in 2017 increased by 4.3%. In the same year, on June 30^{th} the peak traffic load in the EUROCONTROL area reached the highest level of traffic on record, as 35 251 flights were served. The peak day was 23.8% higher than average [23]. Globally, as showed by **Figure 1.1** and **Figure 1.2**, the number of

flights operated every year as well as the annual passenger growth rate have been dramatically increasing. Due to this constantly growing travel demand combined with the economic globalization, up to a two-fold increase in air traffic is projected within the next 20 years [21]. The continuous growth of the air transportation sector places an ever greater strain on the European aviation system's infrastructure. Up to now, more than €5 billion of direct additional costs have been generated every year by the inefficiencies of the European Air Traffic Management (ATM) system [23]. These extra costs are due to various types of delays: en route delays, route extensions, airports delays, arrivals holding and taxi-out. Furthermore, 8.1 million

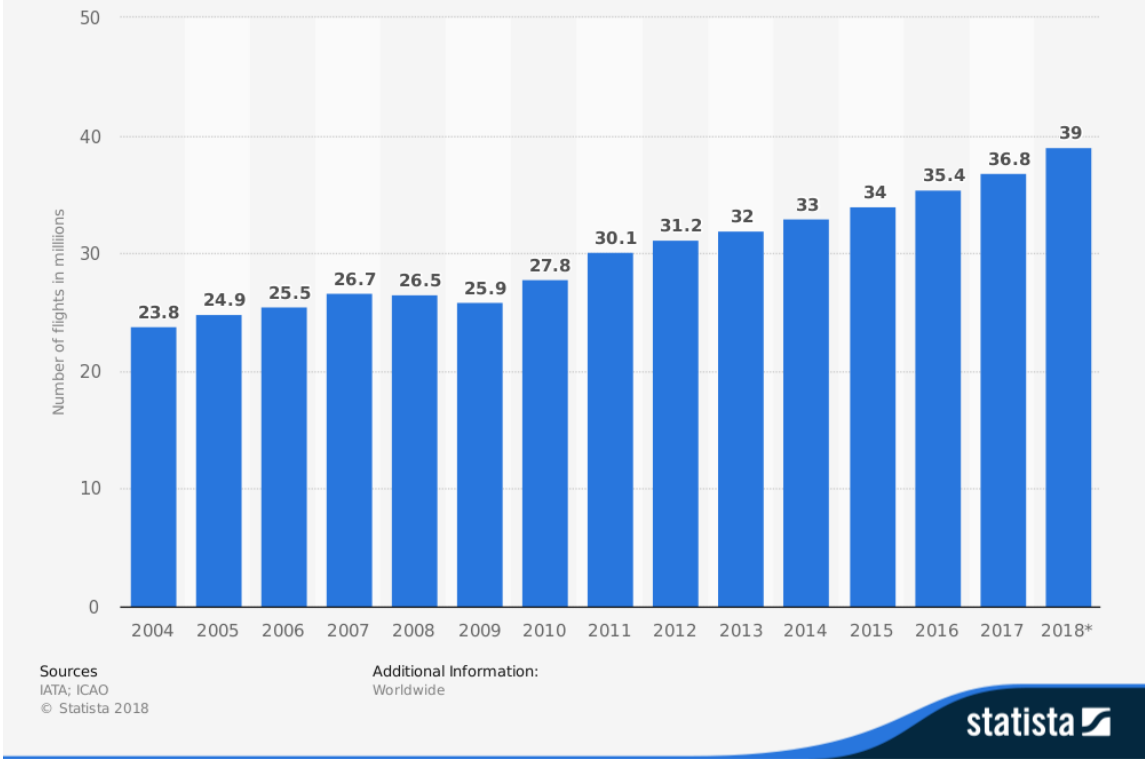


Figure 1.1: number of flights performed by the global airline industry from 2004 to 2018 (in millions) [37]

tonnes of additional CO_2 emission are produced every year and passenger flight delays are estimated to be more than 100 million hours annually (800 million passengers have suffered from 10 minutes delays on average) [23]. Air traffic flow management (ATFM) plays a central role in reducing these costs. The purpose of ATFM is to prevent demand-capacity imbalances by adjusting the flows of aircraft on a domestic or international basis while maximizing the ATM system efficiency. In the next section we give some details on how ATFM is structured and we then define the air traffic flow management problem.

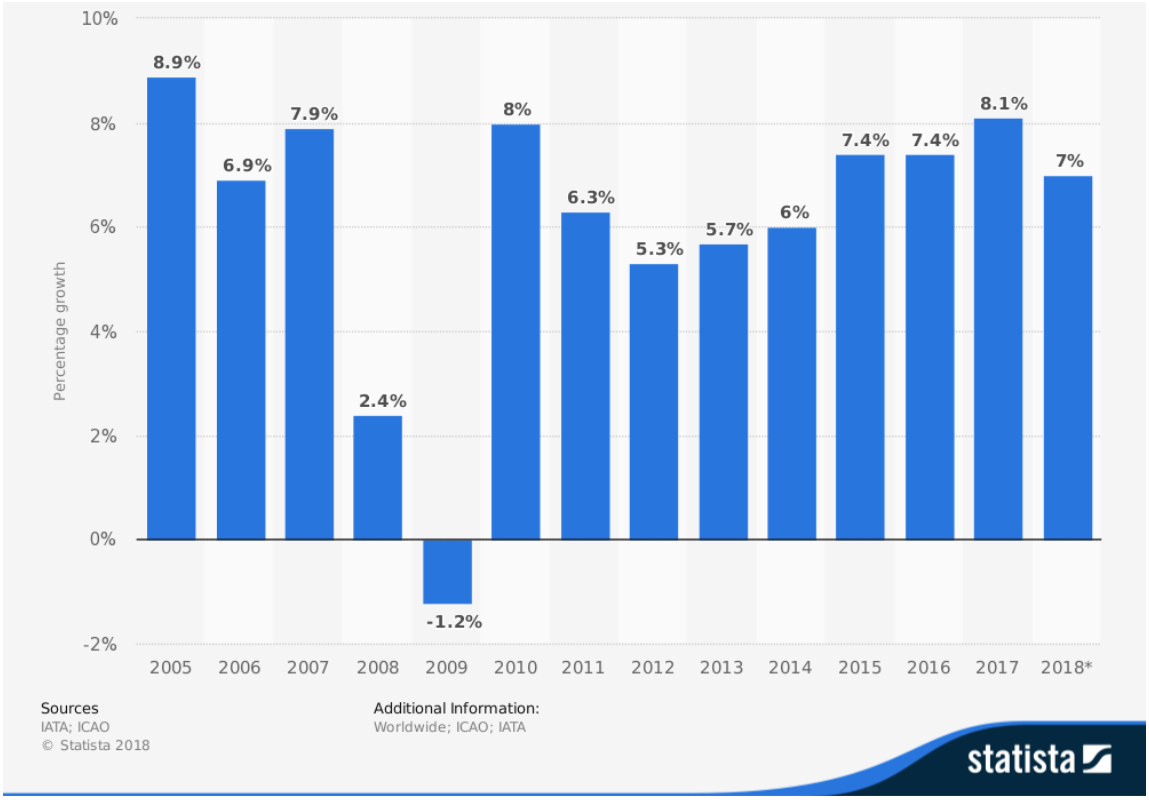


Figure 1.2: annual growth in global air traffic passenger demand from 2005 to 2018 [36]

1.2 Content and contributions

Chapter 1: Introduction In the introduction some general background on the present state of the European air traffic is first provided in order to motivate the present work. A summary of the thesis contents and contribution is also given.

Chapter 2: The ATFM problem After the exposition of how air traffic management network operations are normally conducted, the Air Traffic Flow Management (ATFM) problem is presented within the Air Traffic Management (ATM) scope.

Chapter 3: Framework and main tools In this chapter the reader is first provided with basic historical and theoretical notions on the linear programming (LP) theory. Thereafter, an insight on the LP software development is given, along with a concise description of the main computational tools used in this work.

Chapter 4: State of the art The development of integer linear programming (ILP) models addressing the ATFM problem is presented. Some arguments on ILP models suitability for the ATFM problem are made. A chronological list of the most relevant ILP models in the Operations Research literature for ATFM is finally given.

Chapter 5: A model for ATFM with separation constraints In this core chapter the object of the thesis is presented: a ILP $0-1$ model for the air traffic flow management problem with separation constraints. These constraints constitute the main contribution of the dissertation. First, the key ideas of the dissertation are illustrated. Next, a more detailed insight into the ATFM problem is provided. The model is then described in detail. An additional class of constraints is discussed as well as two classes of valid inequalities. These latter are original results too.

Chapter 6: Model implementation In order to use and test the model a novel program was implemented. In this chapter, details on the implementation of this program are given. The data file form used is described as well as the structure of the program implemented.

Chapter 7: Computational results Some results obtained applying the model on small instances are reported. Comparisons are made with the primitive original model and a preliminary discussion is conducted.

Chapter 8 : In this last chapter final conclusions are drawn. Possible improvements and future developments of the model are discussed.

Appendix In the appendix is reported the code of the Python program written to implement the model along with an example of an input data file.

Chapter 2

The ATFM problem

> "Air traffic management (ATM) considers the trajectory of a manned or unmanned vehicle during all phases of flight and manages the interaction of that trajectory with other trajectories or hazards to achieve the optimum system outcome, with minimal deviation from the user-requested flight trajectory, whenever possible." (ICAO Doc. 9854, §1.9.2)

In this chapter we present the problem addressed in our thesis: the air traffic flow management (ATFM) problem.

2.1 The ATFM

ATFM is "a service established with the objective of contributing to a safe, orderly and expeditious flow of air traffic by ensuring that ATC capacity is utilized to the maximum extent possible, and that the traffic volume is compatible with the capacities declared by the appropriate air trafffic services (ATS) authority" [24]. Its objective is therefore to optimize the air traffic flow according to air traffic control capacity, enabling airlines to operate safe and efficient flights. Air traffic management network operations are based on the ATFM. The Network Manager Operations

Centre (NMOC) permanently monitors the balance between the airspace capacity and the traffic load. Air traffic controllers (ATC) control at operational level the air traffic in real time, [19].

ATFCM activities can be divided into 4 phases:

Strategic phase This phase takes place from months until one week before the actual day of operations. The NMOC supports the air navigation service providers (ANSPs) to evaluate what capacity they will need to provide in each air traffic control centre. A routing scheme is set up.

Pre-tactical phase The phase takes place from one to six days before the actual day of operations. The NMOC guides the definition of a daily plan. It then transmits to ATC and aircraft operators the agreed agenda for the actual day of operations. We are interested in this second phase.

Tactical phase The phase takes place on the day of operations. The NMOC monitors and updates the scheduled plan according to the current situation and the real time air traffic. When aircraft are affected by a regulation, the centre offers alternative solutions to minimize delays.

Operational phase The phase takes place during operations. It mainly involves ATC and aircraft operators, which operate in real time to guarantee to each aircraft safe and efficient flight conditions (e.g. collision avoidance). All the above phases aim at ensuring that the operational phase can be properly carried out.

2.2 The problem

As stated by the quotation at the beginning of this section, the ATM manages trajectories and the relations between them. The purpose of ATM is to achieve “the

optimum system outcome, with minimal deviation from the user-requested flight trajectory". The flight's trajectory gives the information of the departure and arrival airports for that flight, the sequence of airspace sectors crossed by the flight during the journey, along with the time of departure, arrival and sector entrance. It is worth pointing out that by the term trajectory we do not simply mean a 3D space trajectory, but a 4D space-time trajectory. In this sense a trajectory carries the information of where a flight is and when it arrives there. According to this meaning, one of the main goal of ATM is then to avoid delays, which can be seen as deviations from the time component of the requested trajectory. Naturally, the fundamental time deviations are those from the scheduled departure and arrival, since they are the ones implying the main delay costs. In other words, as far as delays costs are concerned, it is not important whether or not there is a delay in the other intermediate stages between take off and landing, but only if the flight departs or arrives on time. Hence the delay costs are mainly related to ground delays at departure and airborne delays at arrival. The second type of delay is normally more expensive, since it implies extra fuel consumption.

One of the main issue that causes delays is the limited capacity of airports and sectors. With the term capacity we mean the ability to provide Air navigation Services with a certain volume of air traffic, maintaining a high level of safety and allowing normal operational conditions, [20]. In particular, we define the sector capacity as "the maximum number of flight entries within one hour that can safely be assigned to sector controllers", [16]. In other words, sector capacity is the maximum number of flights that the sector controllers can process without exceeding a maximum predefined amount of work. The same definition can be given for airports capacity. Some capacity restrictions are then to be imposed in order to allow the ATC to operate safely and efficiently. The capacity of a sector depends of course on its dimensions and geometry, but it also depends on time (e.g. mutable weather conditions). Capacity can then change dynamically in time. Capacity restrictions

are necessary for safety reasons but they are the principal cause of airspace congestion and thus flight delays. Therefore, ATFM problem is to find trajectory deviations that allow to reduce airspace congestion and hence delays. Let us now describe some techniques to achieve this.

Two main strategies are typically used to avoid congestion. The first one is known as *rerouting*. The idea of rerouting is to divert a flight on a different route from the scheduled one, maintaining departure and arrival airports, in order to possibly improve the general state of the air traffic flow. For example, a flight that is scheduled to cross a particularly congested airspace sector may be rerouted in order not to occupy this sector, allowing other flights to cross the sector. The other strategy is commonly known as *ground holding*. The idea of ground holding is to anticipate the delay on the ground at the departure airport. For example, suppose a flight is planned to take off at 8 am and to land at 9 am and suppose that congestion is expected at the arrival airport at 9 am but not at 9:15 am. If the flight departs at the scheduled time it will then have to hold airborne on the destination airport for 15 minutes. If instead we have the flight take off at 8:15 am, it will arrive at destination when there is no congestion. These two techniques can be both seen as a deviation from the scheduled flight trajectory, a space deviation in the first case, a time deviation in the second one. Other two common strategies are *airborne holding* and *speed control*. Airborne holding consists in allowing a flight to hold on airborne before landing at its arrival airport. This strategy is often used, though it implies extra fuel consumption. It allows a flight to depart on time postponing a possible delay due to congestion at the arrival airport, still allowing a punctual flight in case the forecast congestion does not realize. Airborne holding can be seen as the counterpart of ground holding: an optimal balance between these two strategies is a key element in ATFM. For its part, speed control enable flights to change cruise speed in order to reach a certain sector or the arrival airport before or after a forecast congestion (e.g. if a flight is planned to take off at 8 a.m. and to land at 9 a.m. when congestion is forecast, if at 9:10 a.m. no congestion is forecast, the aircraft can extend the flight duration by speed control in order to arrive when no congestion is expected).

Considering the above exposition, we can then state the Air Traffic Flow Management Problem as follows:

> Given a set of flights with initial 4D trajectories, given the airspace configuration and the capacity restrictions determine a set of modified trajectories so that capacity restrictions are satisfied and system efficiency is maximized (e.g. minimized delays).

The purpose of this thesis is to present a mathematical model to address the problem just discussed. In the following chapter we describe the mathematical framework in which the model has been developed: the integer linear programming. The main theoretical and computational tools that we have used are exposed too.

Chapter 3

Framework and main tools

The model presented in this thesis, as well as the models proposed in Operations Research literature that we will present in **Chapter 4**, belongs to the class of integer linear programming (ILP) models, which is an important subclass of linear programming (LP) models. Linear programming is a relatively recent branch of mathematical optimization and "arguably one of the greatest success of computational mathematics in the twentieth century" [7]. It belongs to the applied mathematics field of Operation Research. The objective of LP is to achieve the best outcome (such as maximum profit or lowest cost) in a mathematical model subject to requirements represented by linear relationships (inequalities). We talk about a (mixed) integer linear programming (M)ILP problem whenever the problem solution is bound to be integer (or just a subset of its components is bound to be integer).

Although the problem of solving a system of linear inequalities dates back to 1827, when Jean-Baptiste Joseph Fourier published a method for solving them, the creation of LP as a discipline, together with the recognition of its importance, came only in the 1940's by the seminal work of Dantzig, Koopmans and Von Neumann [18]. Actually already in late 30's a linear programming formulation of a problem was given by the Soviet economist Leonid Kantorovich, who also proposed a method

for solving it. The problem he addressed in his work *Mathematical Methods of Organizing and Planning Production* was to schedule during World War II expenses and returns in order to reduce army costs and to increase enemy losses. In 1947, George Dantzig proposed the simplex algorithm (also known as the simplex method (SM)), which can be applied to any non-integer linear program. Still nowadays, the simplex method remains a fundamental computational instrument in LP. Since then, a lot of progress has been made, both in theoretical and software implementation aspects. Nowadays LP is applied to a plethora of human endeavors and activities. It is widely used in areas as economics, logistics and computational sciences. Many industries use LP as a standard tool. Examples of important application areas include shipping or telecommunication networks, oil refining and blending, stock and bond portfolio selection, airline crew planning and, of course, flight scheduling. Furthermore, LP still plays a central role in pure mathematics fields as graph theory and combinatorics. Some famous and important problem that can be treated through LP and ILP are: the travelling salesman problem, the knapsack problem, the cutting stock problem, set packing, covering and partitioning.

3.1 Integer linear programming

> "Linear programming can be viewed as part of a great revolutionary development which has given mankind the ability to state general goals and to lay out a path of detailed decisions to take in order to "best" achieve these goals when faced with practical situations of great complexity. Our tools for doing this are ways to formulate real-world problems in detailed mathematical terms (models), techniques for solving the models (algorithms), and engines for executing the steps of algorithms (computers and software)." (George Dantzig in [8], 1997)

For this section we mainly referred to [7]. Two other introductory textbooks suggested for the interested reader are [8] and [18]

A *linear program* (LP) (in standard form) is a problem of the form

$$\begin{aligned} \max \quad & cx \\ \text{subject to} \quad & Ax \leqslant b \\ & x \geqslant 0 \end{aligned} \qquad \text{(A)}$$

where $x, c \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$.

The problem is to find a point $\overline{x}$ which maximizes the linear function cx, called the *objective function*. Such a point must satisfy the linear system of inequalities $Ax \leqslant b,\ x \geqslant 0$. In other words a point $\overline{x}$ is a possible solution (a so called *feasible point*) only if it belongs to the so called *feasible region* $\{x \in \mathbb{R}^n |\ Ax \leqslant b,\ x \geqslant 0\}$. Any point $\overline{x}$ which belongs to the feasible region and maximizes the objective function is a solution of the LP problem (A).

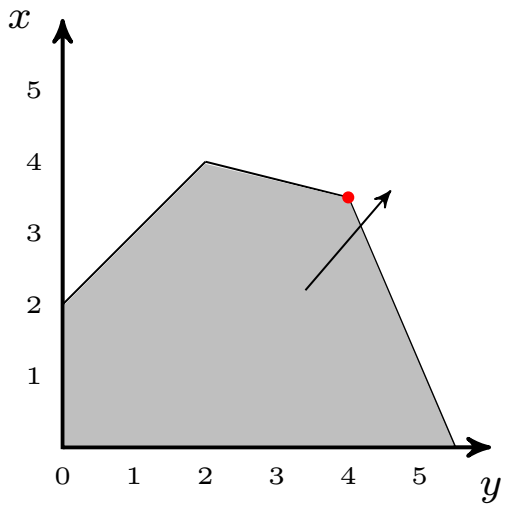


Figure 3.1: A basic linear programming problem

The geometric meaning of the problem is depicted in **Figure 3.1**: it is asked to find the most extreme point along a fixed direction (the small arrow in the picture), determined by the constant vector c, such that this point is contained in the region described by the inequality system $Ax \leqslant b$ (the gray region in the picture). The red dot shows such a point and it therefore represents a (the) solution. A LP does not necessarily admit a solution. In fact, two other cases are possible: the problem can be *infeasible*, meaning that the feasible region is the empty set and therefore no solution

exists, or the problem can be *unbounded*, meaning that its objective function can be made arbitrarily "good" (this can happen when the feasible region is unbounded). One crucial observation is that the geometric nature of the feasible region, which is in general always a polyhedron, ensures that if the problem admits a solution then there exists a solution which is a vertex of the feasible region. To solve this general problem one really famous and successful algorithm is available: the *simplex method*. This algorithm works brilliantly for problems in the form (A), where variables are free to take any real value. It is essentially based on the crucial observation made above, as in order to find the solution it basically enumerates all the vertices of the feasible region. A qualitative step for the problem complexity is taken when the variables involved are integral, for example variables that represent indivisible goods or, more importantly, $(0-1)$ decision variables, where by a decision variable we mean a binary variable that has value 1 if a certain decision is taken, 0 otherwise. In this case the problem is called a *(pure) integer linear program* (ILP) and it takes the following form:

$$\begin{aligned} \max \quad & cx \\ \text{subject to} \quad & Ax \leqslant b \\ & x \geqslant 0 \quad \text{integral} \end{aligned} \qquad \text{(B)}$$

Problems in which only a subset of variables is bound to be integral are called *mixed integer linear programs* (MILP). As already said, from LP to ILP there is a qualitative step in the problems complexity. In fact, the complexity class of LP is P, meaning that there exists a polynomial algorithm to solve it (which is actually not the simplex method but the *Karmarkar's interior point algorithm*), while ILP is in the NP-hard complexity class. A geometric representation of this new problem is given by **Figure 3.2**: in this case the only feasible points, i.e. points that are possible solutions, are represented by the gray dots. Hence, a solution is the integer point that we find as

we move as far as we can along the direction displayed by the small arrow. One such point is the red one. In this case the simplex method is no more utilizable, since the feasible region is no more a continuous area defined by linear inequality, but is a discrete set of points. At first, integer programming might be dismissed as trivial. In fact, one (a computer) could simply enumerate all the feasible integer points and then pick a point that results in the largest value of cx. This is of course possible in theory, but is not practical when the dimensions of the problem are big. Even simple problems are actually impossible to solve using this kind of enumeration approach. Another idea that comes quite naturally to mind is trying to solve the problem with the simplex method without setting the integrality constraints. This idea is known as the problem (integrality constraints) relaxation. After all, one can see that in our example the solution that the simplex algorithm would find (the orange dot) would not be that far from the right one (the red dot). This second naive idea doesn't actually work. In fact, one can show a number of counterexamples in which this approach miserably fails, as the solution of the relaxed problem is arbitrarily distant from the right one. Nevertheless, in the following we describe two main algorithmic idea that are currently heavily used in integer linear programming software that based themselves on this relaxation idea. Summarizing, we want to solve problem (B). We can easily solve problem (A) through the simplex method. Problem (A) and problem (B) are related through the relaxation process (problem (A) is the relaxation of problem (B)). We therefore want to find a way to properly use the simplex method on problem (A) in order to solve problem (B).

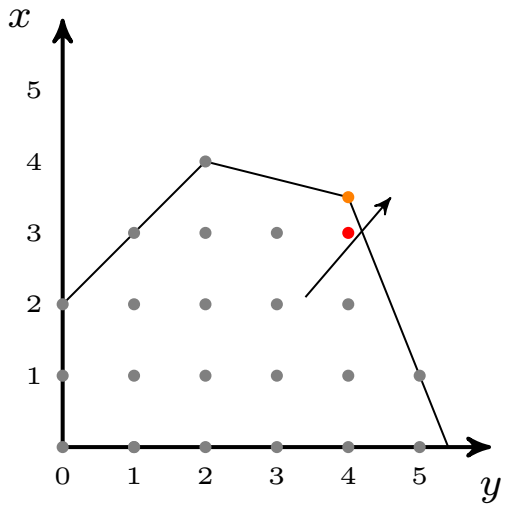


Figure 3.2: A basic integer linear programming problem

3.1.1 The Cutting Planes Method

The idea of cutting-planes method is to iteratively refine a feasible set by means of linear inequalities, termed cuts (or cutting planes). In more detail, cutting plane methods work by solving the linear relaxation of the integer program, i.e. the original problem without integrality constraints, using the simplex method. The obtained solution for the relaxed problem is then tested for being an integer point. Otherwise, then, there must exist a linear inequality separating the current solution from the set of integer points, which are candidate solutions. In fact, as already observed, the solution found with the simplex method must be a vertex. Finding such an inequality is called a *separation problem*. Such an inequality is called a *cut*. Subsequently, we literally split with the selected cut the relaxed region in two splits. Thus the current non-integer solution is contained in one of the two splits, while all the integer points that are candidate solutions are contained in the other split of the relaxed region. The solution of the problem is then contained in the second split. This process is then repeated iteratively until an optimal integer solution is found. We illustrate this method by means of the previous example. We then start using the simplex method to solve the linear relaxation (**Figure 3.1**) of the integer problem (**Figure 3.2**). As a result see **Figure 3.3**: the simplex method gives us the point represented by the red dot. This point is not integral and therefore is not a solution of our integer problem. We then look for a *cut* separating this point from the feasible integer points. One such cut is depicted in **Figure 3.4 (a)**. We now apply the simplex method to the pink region obtained by adding the cut. The solu-

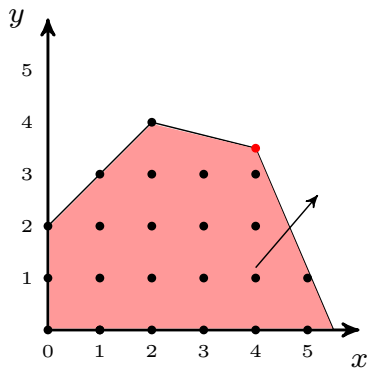
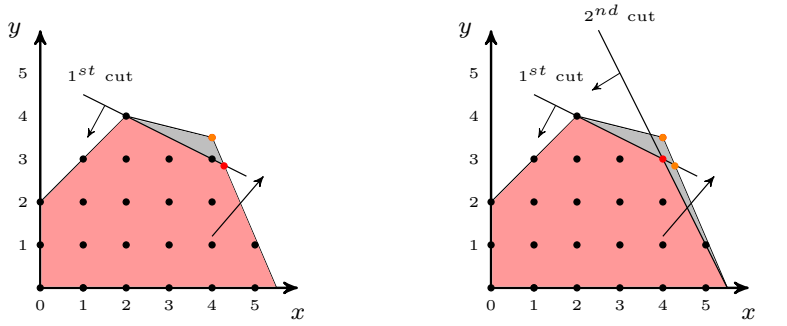


Figure 3.3: First step in CP method

tion to this new problem is again represented by the red dot, while the old solution has been discarded (the orange dot). The new solution is again not integral. It is then necessary to add another cut, separating this solution from the integer feasible points. One such cut is depicted in **Figure 3.4 (b)**. We now apply the simplex method to the new pink region obtained by adding this second cut. The solution to this new problem is represented by the red dot, while the old solution has been discarded (the second orange dot). At this stage we look at the solution obtained and we realize that it is integral. We have thus found the solution of the original integer problem.



(a) Second step in CP method **(b)** Third step in CP method

Figure 3.4

Obviously, it is of crucial importance for the success of this procedure the way we choose the cuts. In fact, the more a cut is tight to the feasible integer points, the better is the cut. In literature a lot of different types of cuts have been proposed, some of which work in general, some others work well only for a specific family of problems. Some well-known cuts families are Gomory's mixed integer cuts and Chvatal's inequalities.

3.1.2 The Branch & Bound Method

We first illustrate this method by means of the previous example. The objective function in this example is $x + y$, maximized in the direction of the small arrow. Let us first consider the solution found for problem (A) with the simplex method (the red dot in **Figure 3.5**). This solution is the point $x_0 = (4, 3.5)$. Since the solution is not integral, at least one of its component is not integral. Let us choose one of these. In our case the only possibility is to take the second component y, with value 3.5. Since we are looking for integral solutions, exactly one of the two following alternatives must necessarily hold: either $y \leqslant 3$ or $y \geqslant 4$. These two inequalities geometrically cut our feasible region into three parts, as depicted in **Figure 3.6 (a)**: a blue region, a gray region and a green region. The gray region does not contain any integer dot, because in this region $3 < y < 4$. We can therefore exclude that the solution of our problem can be found in this region. Hence the solution is contained either in the blue region (which is formed by a single point, the blue dot in **Figure 3.6 (a)**) or in the green one. We now apply the simplex method to both of the regions. In the blue region we find the point $x_1 = (2, 4)$ (the blue dot) which is an integral point in which the objective function takes value 6. In the green region we find the point $x_2 = (4.1, 4)$ (the green dot) with value 8.2. Since x_1 is integral, it may be the solution. However, it is possible to find another integer point in the green region with a value better than 6. We know that because we found an upper bound in the green region, the value of point x_2 found with the simplex method, of 8.2, which is greater than 6. If the value taken by x_2 was smaller

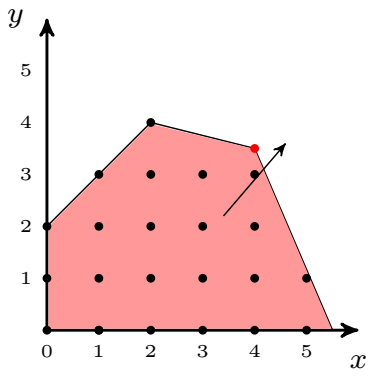
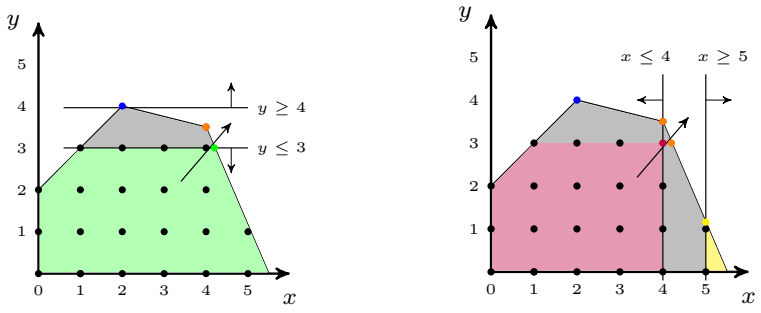


Figure 3.5: First step in B&B

than 6, x_1 would be our solution, because we would not have any chance to find any point in the green region with a value higher than 6. Therefore at this point we iterate the process: since x_2 is not integral, one of its components is not integral. In our case this component is the x one, which values 4.1. Like before, exactly one of the following must hold: either $x \leqslant 4$ or $x \geqslant 5$. These two inequalities geometrically cut the green region again into three parts, as depicted in **Figure 3.6 (b)**: a purple region, a yellow region and the gray region in-between.



(a) Second step in B&B **(b)** Third step in B&B

Figure 3.6

The gray region does not contain any integer point, because in this region $4 < x < 5$. We can therefore exclude that the solution of our problem can be found in this region. Hence the solution is contained either in the purple region or in the yellow one. We now apply the simplex method to both of the regions. In the purple region we find the point $x_3 = (4, 3)$ (the purple dot) which is an integral point in which the objective function takes value 7. In the yellow region we find the point $x_4 = (5, 1.1)$ (the yellow dot) with value 6.1. Since x_3 is integral, it may be the solution. As the value taken by x_4 is smaller than 7 and it is an upper bound for the value of all the points contained in the yellow region, there is no hope to find a integral point in the yellow region with a value better than 7. Hence x_1 is our solution. The Branch and Bound algorithm follows closely the procedure that we

have just described. The name itself tells a lot about how the method works: an alternation between branching phases, in which a feasible region gets split into two feasible subregions; and bounding phases, in which a linear program is solved using the simplex method, finding in this way an upper bound for the region taken into account. The reader familiar with algorithms can easily recognize the binary tree structure of the algorithm, as the *root* consists in the initial linear relaxation of the original problem and every branching phase generates two leaves.

3.1.3 The Branch & Cut Method

In the Branch and Bound method, the tightness of the upper bound is essential for the efficiency of the algorithm. Applying the cutting plane method to the subproblems solved during the Branch and Bound allows to calculate tighter upper bounds. This idea directly leads to the Branch and Cut method, which is "currently the most successful method for solving integer programs" [7]. It works by inserting a cut step before the branching step during the Branch and Bound method. Whether to add new cuts or not before branching is usually decided empirically on the basis of the success of previously added cuts. Typically, various cuts are added at the root node, the linear relaxation of the original integer problem, while fewer or no cuts may be added in the next steps.

3.2 Software framework and tools

In this section we first give some historical background on the development of linear programming software, from the origins until nowadays. Secondly, we illustrate the software programs that we have used.

3.2.1 LP software development

In the following we expose a brief history in the development of LP software (for more historical insight see [9]):

1947 George Dantzig proposes the Simplex Method. The first application of the simplex algorithm to the solution of a non-trivial LP of a 21 constraint, 77 variable instance. It is reported that the total computation time was 120 man-days.

1953 The first machine implementation of the simplex algorithm used a device known as a Card Programmable Calculator (CPC). It was capable of handling LPs with up to 45 constraints and 70 variables, with a total computation time of about 8 hours.

1954–55 SM was improved thanks to new algorithms and re-implemented on an IBM 701, IBM's first real "scientific computer". This implementation could handle LPs with 101 constraints and was used in large computations on an economic model, one of the first real applications of the SM.

1955–56 A new implementation was conceived for the IBM 704. This code was able to handle LPs with up to 255 constraints. It was the first program to be distributed for a wider audience. Later it was improved to handle 512 constraints. In 1956, LP/90 code was developed for the IBM 7090, it was able to handle up to 1024 constraints. LP started being used in the oil industry.

1963–64 LP/90/94 was released for the IBM 7094. It reached the top of first-generation of LP codes. It was used to re-locate factories in Europe by Philips Petroleum, to select ships and transport aircraft to support military deployment by UK, to choose investments on refinery infrastructure by British Petroleum and to select coal mines for closure by the UK National Coal Board.

1970's - 80's A number of new generation of codes was developed and released (APEX, FMPS, MINOS, MPSX, MPS III, XMP, UMPIRE). New methods and ideas came up, as well as the implementation of previous theoretical results, as the dual simplex algorithm, which could not be implemented on the older class of computers.

1979 Leonid Khachiyan showed that LPs could be solved in polynomial time. It was a fundamental theoretical advance, though not unexpected, mainly because of its important implications in the theory of combinatorial optimization. Khachiyan also studied an algorithm known as the *Ellipsoid Method*. Even though this algorithm was proven to be polynomial, it had scarce practical impact, since variants of the SM and the Karmarkar's interior point algorithm are in practice much faster.

1980 IBM introduced the popular personal computer (PC) series IBM 5150. Though PCs were not new at that time, the release of the IBM PC marked the beginning of the business applications of PCs. It was realized that PCs could be used as platforms for the development of practical LP and MIP codes.

1984 Narendra Karmarkar demonstrated a polynomial-time bound for LP that was far better than the bounds for Khachiyan's method and, more importantly, it also corresponded to a computational approach that was applicable in practice, known as the *Karmarkar's interior point algorithm*.

1988 The first version of CPLEX was released by Robert E. Bixby.

1990's Important advances occurred during this period, as the utilization of the dual simplex algorithm as a general purpose solver (not just used in B&B algorithms), the development of dual steepest-edge algorithms and the improvement of linear algebra in the application of SM for large, sparse models.

2000's - modern times R. Bixby has conducted some computational experiments, showing that from 1988 to 2004 the average speed of at least one LP code (namely CPLEX [30]) improved by a factor of roughly 3300. This improvement is independent of any machine effects and is at any rate greater than the improvements in the speed of computing machines in the same period. Combining it with the machines' speed improvements, the resulting factor exceeds six orders of magnitude [9].

Algorithmic improvement (machine independent)	
Best of barrier, primal simplex, and dual simplex:	$\times 3300$
Machine improvement:	$\times 1600$
Total improvement ($3300 \cdot 1600$):	$\times 5\,280\,000$

Other relevant modern software are Gurobi [29], released in 2009 by Gu, Rothberg and Bixby, and FICO Xpress [27], originally developed in 1983 and acquired by FICO in 2008. Gurobi LP solver make use of primal and dual SM, parallel barrier algorithm with crossover, concurrent optimization and sifting algorithm; its MIP uses deterministic, parallel branch and cut, non-traditional tree-of-trees search, multiple default heuristics, solution improvement, cutting planes and symmetry detection [28]. FICO Xpress features techniques as primal and dual SM, Newton barrier algorithm, branch and bound, variable and node selection, cutting planes, integer preprocessing, rounding heuristics [26]. These are just two relevant examples, but the software for LP and (M)ILP now available are actually a great many.

For our purposes, we used the IBM software IBM ILOG CPLEX Optimization Studio (version 12.6.0) through its Python (version 2.7) interface. We now spend some words on the two main instruments that we have used to implement our model (for details on the implementation of our model, see **Chapter 6**).

3.2.2 IBM ILOG CPLEX Optimization Studio

IBM ILOG CPLEX Optimization Studio (informally named CPLEX for short) is an optimization software package. CPLEX is named after the simplex method. CPLEX is implemented in the C programming language, even though nowadays it also provides additional algorithms of mathematical programming and it offers different interfaces towards other environment and programming languages besides C, like C++, C#, Java and Python. It was originally developed by Robert E. Bixby and it has been commercialized since 1988 by CPLEX Optimization Inc. In 1997 it was acquired by ILOG and ILOG was in turn acquired by IBM in 2009. IBM currently maintains and develops CPLEX. Among others, IBM ILOG CPLEX Optimizer can solve integer programming problems and very large linear programming problems. CPLEX is provided with a variety of techniques and algorithms as LP solvers (which use pre-processing, algebra for sparse systems, primal and dual methods, techniques to avoid degeneracy and numerical difficulties), a wide classes of cutting planes, heuristics (such as node heuristics and polishing) and parallelization (such as B&B search).

3.2.3 Python

Python is an open source programming language for general purpose programming. It was created by the dutch programmer Guido van Rossum and first released in 1991. It is named after the British surreal comedy group Monty Python, whom van Rossum was a big fan of. Python has a design philosophy that emphasizes code

readability, clearness and agility. Since 2003, Python has constanly ranked in the top ten most popular programming languages in the TIOBE Programming Community Index and since January 2018 it has been the fourth most popular language behind Java, C, and C++. It was selected Programming Language of the Year in 2007 and 2010. Some well-known large organizations that use Python are Wikipedia, Google, Yahoo!, CERN, NASA, Facebook, Amazon, Instagram, Spotify.

Chapter 4

State of the art

There are different approaches to the ATFM problem and various type of models to mathematically describe it. Models may differ in many aspects. For example some models are more detailed than others, some are deterministic while others are stochastic, some are static while some others are dynamic. Basically, depending on the model, different features and characteristics of the problem are captured. Among the various model types, ILP models started being used to address the ATFM problem as defined in **Chapter 2** in the 1990's and they developed a lot during the last 30 years. The model proposed in our thesis belongs to this family of models. Compared to other types of models, typical characteristics of ILP models are: determinism, problems discretization approach, standard and efficient algorithms availability (as seen in **Section 3.1**), easy generalization, model readability, decision problems suitability. Discretization is in fact a feature shared by all the models described in the next section. One main characteristic of these models is that time is discretized in time periods. This highlights the non-detailed nature of these models, which capture instead the general and macroscopic behaviour of air traffic flow.

4.1 Relevant ILP models for ATFM

The following chronological list reports some of the most important articles published on this subject, all proposing various $0-1$ linear programming models, that is linear programs with decision variables.

Helme (1992) [13]: In this seminal article Marcia P. Helme proposes a linear program to minimize delays through ground holding, allowing for the possibility of an airborne delay. The trajectories are then a given data input and an aircraft can deviate from them only in time. By that, we mean that the only decisions which can be made are: 1) whether or not holding a flight to the ground at the departure airport and if so decide for how much time; 2) whether or not holding a flight airborne over the arrival airport and if so decide for how much time. It is assumed that no speed control during the flight is possible.

Bertsimas & Stock Patterson (1998) [5]: This second paper introduces the possibility of speed control during the flight. Furthermore, it takes into account continued flights, i.e. flights that are continued by other flights. This work by Dimitris Bertsimas and Sarah Stock Patterson has been the start point of our thesis. One major contributions of this paper is the introduction of a new type of variables which has three main benefits: these variables clearly and directly capture some constraints; they define constraints that are facets of the convex hull of solutions; the relaxation of the linear program defined through them is almost always integral. We shall in fact use the same type of variables, defined in **Section 5.4**. This model is entirely reported in the next **Section 4.2**.

Bertsimas & Stock Patterson (2000) [6]: This model, actually tested only on small instances, introduced the possibility for a flight to reroute, deciding which possible flight routes configuration minimizes the total delay costs. Speed control is assumed possible.

Bertsimas, Lulli & Odoni (2011) [4]: In this article is proposed a model including rerouting, speed control, ground and airborne holding and stakeholders fairness (that is, the output flight trajectories configuration should not particularly penalize a stakeholder). It is to some extent an improvement of Bertsimas-Stock Patterson 2000's model, as it was proved able to solve bigger instances. This paper, along with the Bertsimas-Stock Patterson 1998's article, has been the other fundamental source for the present thesis.

Augustin, Alonso-Ayuso, Escudero (2012) [1, 2]: This paper is divided into two parts. The first one presents a mixed 0–1 model which allows flight cancellation and rerouting. It considers several objective functions to minimize such the number of flights exceeding a given time delay, penalization of alternative routes to the scheduled one, time unit delay cost to arrive at intermediate air sectors and penalization for advancing arrival to the sectors and airports over the schedule. It is able to treat large-scale instances. The second part presents a stochastic variation of the previous model, in which uncertainty is introduced. (Note that all the previous models are deterministic).

Fomeni, Lulli & Zografos (2017) [11]: This paper presents a model that contributes to the optimization and optimum configuration of trajectory based operations (TBO). TBO is the concept of improving throughput, flight efficiency, flight times, and schedule predictability through better prediction and coordination of aircraft trajectories [35]. A 0-1 integer programming model is developed with the aim to assign a 4D-trajectory to each flight with the purpose to optimize the efficiency of the ATM system. The model considers the preferred 4D-trajectory of all the flights in the pre-tactical phase and outputs an optimal 4D-trajectory for each flight. These output trajectories are obtained by minimizing the space-time deviation from the original preferred trajectories. The TBO concept implies that these 4D-trajectory are to be shared and negotiated with other stakeholders and subsequently managed throughout the flight. The novelty of this model is that it both considers complete 4D-trajectories for

each flight as well as the preferences and priorities of the ATM stakeholders. In order to describe 4D trajectories, the information of flight level is included in the decision variables.

4.2 The Bertsimas & Stock Patterson 1998 model

This model [5] is at the very basis of our work. We entirely present it in this section. For a deeper understanding of it we strongly suggest to read **Chapter 5** first. The precise form of the ATFM problem addressed by this model is the following: given a user-requested configuration of departure, arrivals and flight trajectories (which in this case are a sequence of sectors along with a scheduled departure and arrival times), find a trajectory configuration which minimizes flight delays, both on the ground and on airborne, while respecting airspace capacity restrictions. Allowed strategies are: ground holding, airborne holding, speed control. In other words, the objective of the problem is to decide how much each flight is going to be held on the ground and in the air in order to minimize the total delay cost. The models considers a set a of flights $\mathcal{F}$, a set of airports $\mathcal{K}$, a set of sectors $\mathcal{J}$, a set of time periods $\mathcal{T}$, a set of pairs of flights that are continued $\mathcal{C} = \{(f, f') : \ f' \text{ is continued by flight } f\}$. The other problem data are given with the following notation:

$N_f \quad = \quad$ number of sectors in flight f's path

$$P(f,i) \quad = \quad \begin{cases} \text{the departure airport,} & \text{if } \ i = 1 \\ \text{the } (i-1)^{st} \text{ sector in flight } f\text{'s path,} & \text{if } \ 1 < i < N_f \\ \text{the arrival airport,} & \text{if } \ i = N_f \end{cases}$$

$P_f \quad = \quad \{P(f,i) : 1 \leqslant i \leqslant N_f\}$

$D_k(t) \quad = \quad$ departure capacity of airport k at time t

$A_k(t)$ = arrival capacity of airport k at time t

$S_j(t)$ = capacity of sector j at time t

d_f = scheduled departure time of flight f

r_f = scheduled arrival time of flight f

I_j^f = number of time units that flight f must spend to cross sector j

$\underline{T}_j^f$ = first feasible times for flight f to arrive to sector j

$\overline{T}_j^f$ = last feasible times for flight f to arrive to sector j

c_g^f = cost of holding flight f on the ground for one unit of time

c_a^f = cost of holding flight f in the air for one unit of time

One of the main innovations of this model is the proposal of a new form of decision variable:

$$x_j^f(t) = \begin{cases} 1 & \text{if flight } f \text{ has reached sector } j \textbf{ by} \text{ time } t \\ 0 & \text{otherwise} \end{cases}$$

The mentioned novelty resides in the use of the proposition **by** instead of **at**, as in previous models. This is critical to the understanding of the formulation. We will spend some more words on this, but let us first present the model:

$$\min \sum_{f \in \mathcal{F}} \Big[\; (c_g^f - c_a^f) \sum_{t \in T_k^f,\ k = P(f,1)} t(x_k^f(t) - x_k^f(t-1)) \\ + \quad c_a^f \sum_{t \in T_k^f,\ k = P(f,N_f)} t(x_k^f(t) - x_k^f(t-1)) \\ + \quad (c_a^f - c_g^f) d_f - c_a^f r_f \; \Big]$$

subject to

$$\sum_{\substack{f \,:\, P(f,1)=k, \\ t \in T_k^f}} (x_k^f(t) - x_k^f(t-1)) \leqslant D_k(t) \qquad \forall\ k \in \mathcal{K},\ t \in \mathcal{T} \tag{1}$$

$$\sum_{\substack{f \,:\, P(f,N_f)=k, \\ t \in T_k^f}} (x_k^f(t) - x_k^f(t-1)) \leqslant A_k(t) \qquad \forall\ k \in \mathcal{K},\ t \in \mathcal{T} \tag{2}$$

$$\sum_{\substack{f \,:\, P(f,i)=j,\ P(f,i+1)=j', \\ i<N_f,\ t \in T_j^f}} (x_j^f(t) - x_{j'}^f(t)) \leqslant S_j(t) \qquad \forall\ j \in \mathcal{J},\ t \in \mathcal{T} \tag{3}$$

$$x_{j'}^f(t + I_j^f) - x_j^f(t) \leqslant 0 \qquad \begin{cases} \forall f \in \mathcal{F},\ t \in T_j^f, \\ j = P(f,i), \\ j' = P(f,i+1) \end{cases} \tag{4}$$

$$x_k^f(t) - x_k^{f'}(t - s_{f'}) \leqslant 0 \qquad \begin{cases} \forall (f', f) \in \mathcal{C},\ t \in T_k^f, \\ k = P(f,1) = P(f', N_f) \end{cases} \tag{5}$$

$$x_j^f(t-1) - x_j^f(t) \leqslant 0 \qquad \forall f \in \mathcal{F},\ j \in P_f,\ t \in T_k^f \tag{6}$$

$$x_{j,l}^f(t) \in \{0,1\} \qquad \forall\ f \in \mathcal{F}\ j \in P_f,\ t \in T_j^f \tag{7}$$

The objective function calculates the total delay cost in function of the trajectories configuration. We explicit the derivation of it in **Section 5.6** (it is exactely the same objective function that we used).

Constraints (1), (2) and (3) take into account the capacities of airports (departure and arrival) and of sectors.

Constraints (4) represent connectivity between sectors. They state that that if a flight arrives at sector j' by time $t + I_j^f$, then it must have arrived at the subsequent sector j by time t.

Constraints (6) represents connectivity between airports for continued flights. If (f', f) is a continued flight, if f departs from airport k by time t, then flight f' must have arrived at airport k by time $t - s_{f'}$, where s_f is the turnaround time, which takes into account the time needed to clean, refuel, unload and load and prepare the aircraft for the next flight.

Constraints (7) represent connectivity in time: if a flight f has arrived at sector j by time t, then $x_j^f(t)$ has to have value 1 for all the later times period $s \geqslant t$.

The importance of the utilization of variables $x_j^f(t)$ comes from the following considerations and facts: they clearly and nicely capture the three types of connectivity (between sectors, between aiports and connectivity in time); constraints (5), (6) and (7) are facets of the convex hull of the feasible region defined by the model constraints; the utilization of these variables in the formulation is probably, according to the authors, the reason why the linear relaxation of the feasible region is almost always integral. This fact highlights the high importance of the introduction and usage if these variables. We will in fact use the same variables type, as the reader can see in **Section 5.4**

Chapter 5

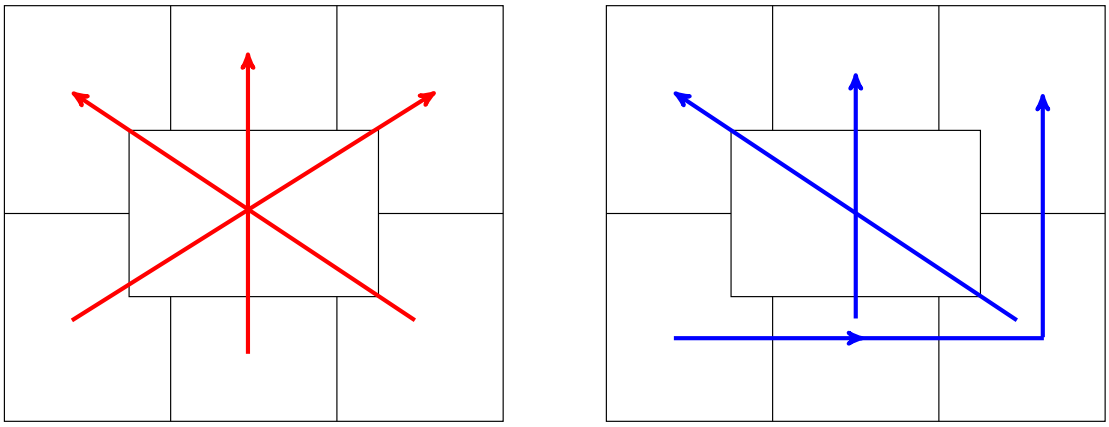
A model for ATFM with separation constraints

On the basis of [5], we build an integer $0-1$ programming model in order to treat and solve the Air Traffic Flow Management Problem presented in **Chapter 2**. Our purpose is to show that introducing separation constraints allows us to relax the sector capacity restrictions, obtaining a better solution still remaining under well controlled safety conditions.

5.1 Key ideas

At the base of our thesis it is the following key observation: the analysis of the data on historical flight trajectories available from the data repository DDR2 [25] shows that there exists discrepancy between sector capacity and approved initial flight plans. In other words, it seems that is still safe to allow more flights to cross a sector than the declared number. In fact, it has been observed that up to 20% of sectors present up to a 50% capacity violation, for example in one sector out of five, though the sector can nominally support e.g. 10 flights every hour, it can in practice support up to 15 flights. Of course in such extreme cases traffic controllers are not

operating comfortably, but they are still able to control the traffic safely. Obviously, an increase in the sector capacities implies an improvement of the air traffic flow. Our proposal is then to use "observed" capacities instead of nominal ones, while still maintaining safety. In order to ensure safety despite the increase of capacity we introduce some kind of separation constraints between the flights. In other words our goal is to describe an ATFM model with increased capacity on the one hand and (simple) operational separation constraints on the other hand. Let us illustrate this idea using the following toy example



(a) Infeasibile configuration **(b)** Feasible configuration

Figure 5.1

Suppose an airspace sectorisation consists of 6 sectors as in **Figure 5.1** and assume that 3 flights have as preferred trajectories those depicted in figure **Figure 5.1a** and assume that these flights are scheduled to cross the central sector at the same time. If the nominal capacity of the sector in the center is 2, such configuration, which is the one requested by the airspace users, is infeasible (that is unviable), since the capacity of that sector would be violated. One of the three flights would then be forced to reroute (see **Figure 5.1b**). This new configuration would then be feasible but it would be more costly, since it would imply the delay of the rerouted flight.

Our point is that the potential capacity of the central sector is in actual fact higher than its nominal capacity. The main reason for that is that the vertical dimension of the sector allows three "separated" flight levels. The idea is then to allow a capacity increase by restricting the flights to travel on well separated flight levels. This would make the configuration in **Figure 5.1a** requested by the users feasible.

The key idea of our work is to compensate the reduction of safety resulting from an increase of the nominal capacities, which as showed in the above toy example should lower the entity of delays, with the imposition of such separation restrictions. In fact, the main difference that our model presents with the models discussed in **Section 4** is the imposition of vertical and horizontal separation constraints, by which we can control in more detail the distance between flights. Rerouting is not possible in our model; we try to accommodate every user with its preferred route. Hence, the only decisions to be made for a given flight trajectory are: when the flight departs and how long it is held airborne over the arrival airport; at which level the flight crosses each sector on its path. Moreover, we do not allow speed control. Instead, the possibilities of both ground and airborne holding are modelled. One last observation is worth being made: there is an analogy between choosing a route and choosing a level, since they both imply a change of direction, horizontal in the first case, vertical in the second. In other words, the possibility to change level during a flight can be seen to some extent as a vertical rerouting. We exploited this idea to reuse some concepts contained in the past papers concerning rerouting (in particular [4]), especially to find the two classes of valid inequalities presented in **Section 5.9**. Furthermore, the need for the introduction of two different type of variables in our model comes from analogous reasons to those of [4]. This analogy arises again from the similarity between the concepts of rerouting and levels distribution.

In accordance to the ideas that we just exposed, it is necessary to describe in more detail the definition of the ATFM problem, introducing concepts that will be formulated in our model.

5.2 An insight into the ATFM problem

The ATFM problem is characterized by various elements, that we describe below.

5.2.1 Sectors

The airspace is virtually divided into Flight Information Regions (FIRs). Each FIR is managed by a controlling authority, responsible for ensuring ATS to every aircraft flying within it. To manage the airspace in a FIR, ANSPs divide it into sectors. These sectors are like 3D puzzle pieces with differing heights and sizes [32]. As an example see **Figure 5.2**, depicting the flight information region of Milan. The boundaries of sectors are formed by a number of factors, many of them are ATS-oriented, others defined by international relations [33]. The total number of airspace sectors in Europe is 679 [38]. Moreover, every airport is contained in a specific sector. For our purposes, flight routes can be seen as a sequence of sectors, as we are not interested to capture flight trajectories in details (as already noted in **Chapter 4** we are only interested in the general behaviour of air traffic flow). One of the ATFM problem data is then a set of flights, along with their pre-determined routes, that is for each flight an ordered sequence of the type (ABC, a, b, c, d, XYZ) is assigned, where ABC, XYZ are airports (identified by their IATA code) and a, b, c, d are airspace sectors. It is finally worth pointing out again that sectors are not bi-dimensional and they can actually overlap.

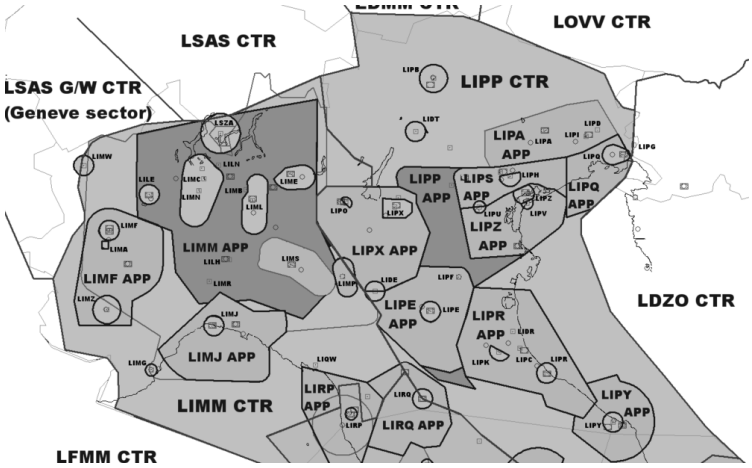


Figure 5.2: The Milan Flight Information Region [39]

5.2.2 Capacity

As we have seen in **Chapter 2**, capacity is a crucial concept in ATFM. In that chapter we defined sector capacity as the maximum number of flights entering a sector during one hour that sector controllers can process without exceeding a maximum predefined amount of work. This definition is dynamic, meaning that capacity is time-dependent. In our model we shall discretize a long time period into smaller time windows, hence it is natural for us to redefine capacity as a maximum number of flights entering a sector during one single *time period*, which is usually shorter than one hour (we shall consider 15-minutes time periods).

5.2.3 Flight levels

Another important characteristic of the airspace structure used by ATM is its subdivision into vertical levels. These levels, that are actually defined by their *pressure altitude*, that is by atmospheric pressure and not by standard altitude, can be seen as the highways used by flights. Usually, each level is separated from the immediate lower and higher levels by 500 ft (approximately 150 m). One key safety standard is the separation between flights. ICAO specifies the minimum vertical separation between flights as 1000 ft (300 m) below FL290 (the flight level at a pressure altitude of 29 000 ft) and 2000 ft (600 m) above FL290 [34]. In our model we take into account this level subdivision, since we want to use it to vertically distribute flights. Flight level is one of the trajectory element that we want to include in the problem modelling. Hence we shall think of a trajectory as a sequence of sectors, in which for each sector the flight level of the trajectory is specified, as well as the time of entrance of the considered flight in that sector. Given a sector and a flight, not every level is allowed for the flight. For example some levels are allowed only for some specific aircraft classes. If a flight is allowed to fly at a certain level in a certain sector, we say that the level is *feasible* for that flight in that sector. Hence, included as a problem data, a set of feasible levels is given for each sector and for each flight.

Furthermore, when a flight passes from a sector to the next one it is allowed to change its flight level only if the level variation is under the threshold of a given maximum level variation. This maximum level variation is assumed to be sector dependent and it is again a given data of our problem. Let us now spend some words on another key concept of our model: flight separation.

5.2.4 Separation

For safety reasons, aircraft trajectories must be horizontally and vertically separated. As already stated in the previous subsection, a vertical separation, meant as a level separation, must be always maintained. This separation between flights usually depends on the directions of the flight taken into account. In our model we shall then set vertical separation according to directions. The horizontal separation is instead usually calculated on a time basis, so for example we say that two flights are separated by 15 minutes if, once they have reported their position, it takes 15 minutes for a flight to reach the other one. Of course, the required amount of horizontal time separation depends on the angle between the two flight's trajectory. For example, if they are travelling on the same route with opposite directions the horizontal distance between them will be rapidly decreasing, until a possible collision occurs. In this case it is typically required that the two flights travel maintaining a sufficiently high level of separation. Summing up, we assume that, in order to guarantee horizontal separation, it is necessary that two flights entering a sector from concordant directions must be separated by a suitable time distance. Moreover, we assume that two flights entering a sector from non-concordant directions must fly at suitably different flight levels. The mathematical translation, that is the formalization, of these separation requirements is one of the core elements of the present thesis.

5.2.5 Time

A set of time periods is given as a problem data too. In fact, as already mentioned at the end of **Chapter 4**, time discretization of a bigger time period (for example a

5 hours period subdivided into 20 15-minutes periods) is a typical feature shared by all the integer programming models covered in **Chapter 4**. We outline once again that this discretization allows us on the one hand to use ILP models and on the other hand means that our model can't really capture detailed aspect of ATFM. Since we are only interested in macroscopic aspects, this is not a problem.

5.2.6 Delay

Each flight has a scheduled departure time and a scheduled arrival time. Each flight has a fixed maximum ground delay, meaning that to each flight a time period subsequent to the scheduled departure time is assigned by which the flight must take off. The amount of time required for a flight to entirely cross a sector is another problem given information. This amount of time can change depending on the flight and the sector. For the sake of simplicity, in our model we will assume that speed control is not possible during the flight. In other words the departure time determines the time in which the flight crosses each sector in its path, except for the landing time, since we will permit airborne holding over the arrival airport. Therefore, once a flight has departed, the departure time determines the time in which the flight reaches the airspace sector to which the arrival airport belongs. It does not determine the landing time, since every flight is allowed to hold airborne before landing. To each flight is assigned a time period by which it must land at the arrival airport. From the information above can be easily calculated the feasible times for a flight to reach a certain sector. This times are exactly the feasible departure times plus the time that a flight must spend in each sector preceding the sector under consideration. All the above gives us a description of a specific air space on which a set of flights travels. One would like all the flights to take off and to land at the scheduled time, so that no delay occurs. However, due to congestion at the arrival airport, in some cases a flight has to hold airborne before landing. This of course implies some delay and therefore extra costs. For each time period spent in airborne hold each flight has a related cost, which we will refer to as the airborne delay cost and which is a problem given data.

As exhaustively illustrated in **Chapter 2**, one of the most common strategies to avoid this kind of delay is ground holding. Since keeping an aircraft on the ground is cheaper than to keep it on airborne, this can reduce the delay costs, but it still causes delay expenses. We will refer to the cost of holding a flight to the ground for one time period as the flight's ground holding delay cost and this cost is a problem given data.

The objective of the model presented in this chapter is to find a configuration of departures and arrivals which minimizes total delay cost. To this purpose the strategies that we are allowed to use are ground holding, airborne holding and distribution by levels of the air traffic.

5.3 Problem data

The problem data are given with the following notation:

$\mathcal{F} = \{f_1, \ldots, f_i, \ldots, f_{M_\mathcal{F}}\}$ is the set of flight

$\mathcal{K} = \{k_1, \ldots, k_i, \ldots, k_{M_\mathcal{K}}\}$ is the set of airports

$\mathcal{J} = \{j_1, \ldots, j_i, \ldots, j_{M_\mathcal{J}}\}$ is the set of airspace sectors

$\mathcal{L} = \{l^0, \ldots, l^i, \ldots, l^{M_\mathcal{L}}\}$ is the set of airspace levels

$\mathcal{T} = \{t_1, \ldots, t_i, \ldots, t_{M_\mathcal{T}}\}$ is the set of time periods

$\mathcal{A} = \{a_1, \ldots, a_i, \ldots, a_{M_\mathcal{A}}\}$ is the set of airspace edges

N_f = number of sectors in flight f's path

$$P(f,i) = \begin{cases} \text{the departure airport,} & \text{if } i = 1 \\ \text{the } (i-1)^{st} \text{ sector in flight } f\text{'s path,} & \text{if } 1 < i < N_f \\ \text{the arrival airport,} & \text{if } i = N_f \end{cases}$$

$P_f = \{P(f,i) : 1 \leqslant i \leqslant N_f\}$

$D_k(t)$ = departure capacity of airport k at time t

$A_k(t)$ = arrival capacity of airport k at time t

$S_j(t)$ = capacity of sector j at time t

d_f = scheduled departure time of flight f

r_f = scheduled arrival time of flight f

I_j^f = number of time units that flight f must spend to cross sector j

$\underline{T}_j^f$ = first feasible times for flight f to arrive to sector j

$\overline{T}_j^f$ = last feasible times for flight f to arrive to sector j

T_j^f = set of feasible times for flight f to arrive to sector j = $[\underline{T}_j^f, \overline{T}_j^f]$

$\mathcal{L}_j^f$ = set of feasible flight level for flight f in sector j

δ_j^f = maximum level variation for flight f between sector j and its successor

t_{sep}^C = time (horizontal) separation for potential chasing conflict

l_{sep}^C = level (vertical) separation for potential chasing conflict

t_{sep}^O = time (horizontal) separation for potential opposing conflict

l_{sep}^O = level (vertical) separation for potential opposing conflict

$K^C(a)$ = the set of potential chasing conflicting sectors centered in a

$K^O(a)$ = the set of potential opposing conflicting sectors opposite to a

M_a = maximum number of flights that use an edge in a potential opposing conflict with a

c_g^f = cost of holding flight f on the ground for one unit of time

c_a^f = cost of holding flight f in the air for one unit of time

5.4 Decision variables

We now want to discuss which are the most suitable decision variables to use for our model's purposes. One can think at three different natural variable types, for $f \in \mathcal{F},\ j \in \mathcal{J},\ l \in \mathcal{L},\ t \in \mathcal{T}$ we define (see **Figure 5.3**):

$$w^f_{j,l}(t) = \begin{cases} 1 & \text{if flight } f \text{ reaches sector } j \text{ at level } l \textbf{ at} \text{ time } t \\ 0 & \text{otherwise} \end{cases}$$

$$x^f_{j,l}(t) = \begin{cases} 1 & \text{if flight } f \text{ has reached sector } j \text{ at level } l \textbf{ by} \text{ time } t \\ 0 & \text{otherwise} \end{cases}$$

$$y^f_{j,l}(t) = \begin{cases} 1 & \text{if flight } f \text{ is in sector } j \text{ at level } l \textbf{ at} \text{ time } t \\ 0 & \text{otherwise} \end{cases}$$

In the model presented here we use the last two types of variables. The main reason is that one can easily switch between variables $w^f_{j,l}(t)$ and $x^f_{j,l}(t)$, since as showed below they are in a linear relation. Hence one could express every expression in the model indistinctly through one or the other type. As pointed out by Bertsimas and Stock Patterson (see **Section 4.2**), the main reason why we prefer to use variables $x^f_{j,l}(t)$ is that they allow us to clearly and nicely capture the two types of connectivity constraints: connectivity in space and connectivity in time (see constraints (6a), (6b), (6c) and (9)). We still need additional variables $y^f_{j,l}(t)$ in order to be able to express opposing conflict separation constraints (7b). Unfortunately in this case the relation between the two is not linear (a max operator is involved). In fact, as already mentioned at the end of **Section 5.1**, we face the same problem as in [4].

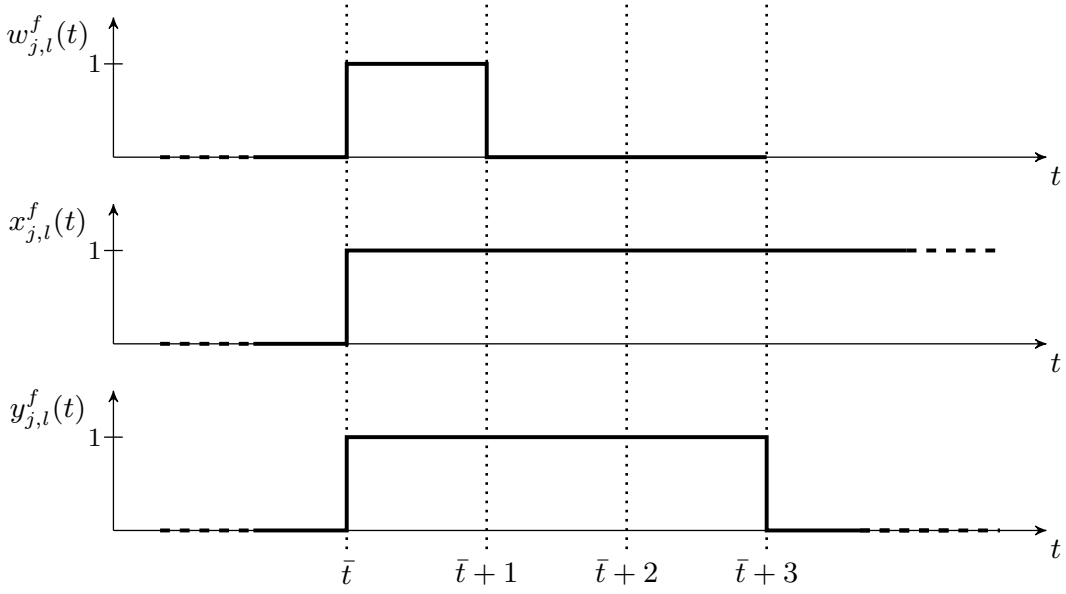


Figure 5.3: a representation of the three variable types

The relations between the three types of variables are the following:

$$\begin{cases} w^f_{j,l}(t) = x^f_{j,l}(t) - x^f_{j,l}(t-1) \\ x^f_{j,l}(t) = \sum_{s \leqslant t} w^f_{j,l}(s) \end{cases}$$

$$\begin{cases} y^f_{j,l}(t) = \max\left\{0,\ x^f_{j,l}(t) - \sum_{\substack{l' \in \mathcal{L}^f_{j'}, \\ j'=succ(j)}} x^f_{j',l'}(t)\right\} \\ x^f_{j,l}(t) = \max\left\{0,\ y^f_{j,l}(t) - y^f_{j,l}(t-1)\right\} \end{cases}$$

$$\begin{cases} y^f_{j,l}(t) = \max\left\{0,\ \sum_{s \leqslant t} w^f_{j,l}(s) - \sum_{\substack{l' \in \mathcal{L}^f_{j'}, r \leqslant t \\ j'=succ(j)}} w^f_{j',l'}(r)\right\} \\ w^f_{j,l}(t) = \max\left\{0,\ y^f_{j,l}(t) - y^f_{j,l}(t-1)\right\} - \max\left\{0,\ y^f_{j,l}(t-1) - y^f_{j,l}(t-2)\right\} \end{cases}$$

5.5 The model

$$\min \sum_{f\in\mathcal{F}} \Big[\; (c_g^f - c_a^f) \sum_{t\in T_k^f,\ k=P(f,1)} t(x_{k,l^0}^f(t) - x_{k,l^0}^f(t-1)) + \; c_a^f \sum_{t\in T_k^f,\ k=P(f,N_f)} t(x_{k,l^0}^f(t) - x_{k,l^0}^f(t-1)) + \; (c_a^f - c_g^f)d_f - c_a^f r_f \; \Big]$$

subject to

$$\sum_{\substack{f\ :\ P(f,1)=k,\\ t\in T_k^f}} (x_{k,l^0}^f(t) - x_{k,l^0}^f(t-1)) \leqslant D_k(t) \qquad \forall\ k\in\mathcal{K},\ t\in\mathcal{T} \tag{1}$$

$$\sum_{\substack{f\ :\ P(f,N_f)=k,\\ t\in T_k^f}} (x_{k,l^0}^f(t) - x_{k,l^0}^f(t-1)) \leqslant A_k(t) \qquad \forall\ k\in\mathcal{K},\ t\in\mathcal{T} \tag{2}$$

$$\sum_{\substack{f\ :\ P(f,i)=j,\ P(f,i+1)=j',\\ i<N_f,\ t\in T_j^f,\ l\in\mathcal{L}_j^f,\ l'\in\mathcal{L}_{j'}^f}} (x_{j,l}^f(t) - x_{j',l'}^f(t)) \leqslant S_j(t) \qquad \forall\ j\in\mathcal{J},\ t\in\mathcal{T} \tag{3}$$

$$y_{j,l}^f(t) \;\geqslant\; x_{j,l}^f(t) \;\; - \sum_{l'\in\mathcal{L}_{j'}^f} x_{j',l'}^f(t) \qquad \begin{cases} \forall\ f\in\mathcal{F},\ 1<i<Nf \\ j=P(f,i),\ j'=P(f,i+1), \\ l\in\mathcal{L}_j^f,\ t\in[\underline{T}_j^f, \overline{T}_j^f + I_j^f] \end{cases} \tag{4}$$

$$\sum_{l\in\mathcal{L}_j^f} x_{j,l}^f(t) \leqslant 1 \qquad \forall\ f\in\mathcal{F},\ j\in P_f,\ t\in T_j^f \tag{5}$$

$$\sum_{l\in\mathcal{L}_j^f} x_{j,l}^f(t) - \sum_{l'\in\mathcal{L}_{j'}^f} x_{j',l'}^f(t+I_j^f) = 0 \qquad \begin{cases} \forall f\in\mathcal{F},\ i<N_f-1, \\ j=P(f,i),\ j'=P(f,i+1), \\ t\in T_j^f \end{cases} \tag{6a}$$

$$x_{j,l}^f(t) \;\; - \sum_{l'\in\mathcal{L}_{j'}^f\cap[l-\delta_j^f,l+\delta_j^f]} x_{j',l'}^f(t+I_j^f) \leqslant 0 \qquad \begin{cases} \forall f\in\mathcal{F},\ i<N_f-1, \\ j=P(f,i),\ j'=P(f,i+1), \\ t\in T_j^f,\ l\in\mathcal{L}_j^f \end{cases} \tag{6b}$$

$$\sum_{l\in\mathcal{L}_j^f\cap[l^0,l^0+\delta_j^f]} x_{j,l}^f(t) \;\; - \;\; x_{k,l^0}^f(t+I_j^f) \geqslant 0 \qquad \begin{cases} \forall f\in\mathcal{F},\ t\in T_j^f, \\ j=P(f,N_f-1), \\ k=P(f,N_f) \end{cases} \tag{6c}$$

$$\sum_{s=t}^{t+t_{sep}^C} \sum_{m=l}^{l+l_{sep}^C} \sum_{k\in K^C(a)} \sum_{\substack{f\,:\,(k,j')\in P_f\\ m\in\mathcal{L}_{j'}^f,\ s\in T_{j'}^f}} \Big(x_{j',m}^f(s) - x_{j',m}^f(s-1)\Big) \leqslant 1 \qquad \begin{cases} \forall\ a=(j,j')\in\mathcal{A}, \\ l\in\mathcal{L},\ t\in\mathcal{T} \end{cases} \tag{7a}$$

$$y_{j',l}^f(t) \;+\; u_{a,l}(t) \leqslant 1 \qquad \begin{cases} \forall\ a=(j,j')\in\mathcal{A}, \\ f\in\mathcal{F} \text{ s.t. } a\in P_f \\ l\in\mathcal{L}_{j'}^f,\ t\in[\underline{T}_{j'}^f, \overline{T}_{j'}^f + I_{j'}^f] \end{cases} \tag{7b}$$

$$\sum_{s=t}^{t+t_{sep}^O} \sum_{m=l-l_{sep}^O}^{l+l_{sep}^O} \sum_{k\in K^O(a)} \sum_{\substack{f'\,:\,(k,j')\in P_{f'}\\ m\in\mathcal{L}_{j'}^{f'},\ s\in T_{j'}^{f'}}} \Big(x_{j',m}^{f'}(s) - x_{j',m}^{f'}(s-1)\Big) \leqslant M_a u_{a,l}(t) \qquad \begin{cases} \forall\ a=(j,j')\in\mathcal{A}, \\ l\in\mathcal{L},\ t\in\mathcal{T} \end{cases} \tag{7c}$$

$$x_{k,0}^f(\overline{T}_k^f) = 1 \qquad \forall\ f\in\mathcal{F},\ k=P(f,N_f) \tag{8}$$

$$x_{j,l}^f(t) - x_{j,l}^f(t-1) \geqslant 0 \qquad \forall\ f\in\mathcal{F},\ j\in P_f,\ l\in\mathcal{L}_j^f,\ t\in T_j^f \tag{9}$$

$$x_{j,l}^f(t),\ y_{j,l}^f(s),\ u_{a,m}(r) \in \{0,1\} \qquad \begin{cases} \forall\ f\in\mathcal{F}\ j\in P_f,\ l\in\mathcal{L}_j^f, \\ t\in T_j^f,\ s\in[\underline{T}_j^f, \overline{T}_j^f + I_j^f] \\ a\in\mathcal{A},\ m\in\mathcal{L},\ r\in\mathcal{T} \end{cases} \tag{10}$$

5.6 The objective function

Our objective is to minimize total delay cost. Let us now construct the objective function which gives the total cost of delays as a function of $x^f_{j,l}(t)$ variables, i.e. trajectories configurations.

The total number of time units that a flight f is held on the ground, let it be g_f, equals the actual departure time minus the scheduled departure time, i.e.,

$$\begin{aligned} g_f &= \sum_{t\in T^f_k, k=P(f,1)} t w^f_{k,l^0}(t) - d_f \\ &= \sum_{t\in T^f_k, k=P(f,1)} t(x^f_{k,l^0}(t) - x^f_{k,l^0}(t-1)) - d_f \end{aligned}$$

The total number of time units that a flight f is held on airborne, let it be a_f, equals the actual arrival time minus the scheduled arrival time minus the amount of time that the flight has been held to the ground, i.e.,

$$\begin{aligned} a_f &= \sum_{t\in T^f_k, k=P(f,N_f)} t w^f_{k,l^0}(t) - r_f - g_f \\ &= \sum_{t\in T^f_k, k=P(f,N_f)} t(x^f_{k,l^0}(t) - x^f_{k,l^0}(t-1)) - r_f - g_f \end{aligned}$$

Using the variables g_f and a_f we can simply express the objective function as follows:

$$\sum_{f\in\mathcal{F}} [c^f_g g_f + c^f_a a_f]$$

Explicitly rewriting the objective function in terms of $x^f_{j,l}(t)$ we obtain the expression:

$$\begin{aligned} \sum_{f\in\mathcal{F}} \Big[& c^f_g \Big(\sum_{t\in T^f_k, k=P(f,1)} t(x^f_{k,l^0}(t) - x^f_{k,l^0}(t-1)) - d_f \Big) \\ & + c^f_a \Big(\sum_{t\in T^f_k, k=P(f,N_f)} t(x^f_{k,l^0}(t) - x^f_{k,l^0}(t-1)) - r_f \\ & - \Big(\sum_{t\in T^f_k, k=P(f,1)} t(x^f_{k,l^0}(t) - x^f_{k,l^0}(t-1)) - d_f \Big) \Big) \Big] \end{aligned}$$

Finally, rearranging terms we can obtain the following formulation of the objective function.

$$\sum_{f\in\mathcal{F}}\Big[\ (c_g^f - c_a^f)\sum_{t\in T_k^f,\ k=P(f,1)} t(x^f_{k,l^0}(t) - x^f_{k,l^0}(t-1)) \\ +\ \ c_a^f \sum_{t\in T_k^f,\ k=P(f,N_f)} t(x^f_{k,l^0}(t) - x^f_{k,l^0}(t-1)) \\ +\ \ (c_a^f - c_g^f)d_f - c_a^f r_f\ \Big]$$

5.7 Model constraints

5.7.1 Capacity constraints

The constraints $(1), (2), (3)$ limit the capacity of departure and arrival airports, and of en-route sectors.

$$\sum_{\substack{f\ :\ P(f,1)=k,\\ t\in T_k^f}} (x^f_{k,l^0}(t) - x^f_{k,l^0}(t-1)) \leqslant D_k(t) \qquad \forall\ k\in\mathcal{K},\ t\in\mathcal{T} \tag{1}$$

$$\sum_{\substack{f\ :\ P(f,N_f)=k,\\ t\in T_k^f}} (x^f_{k,l^0}(t) - x^f_{k,l^0}(t-1)) \leqslant A_k(t) \qquad \forall\ k\in\mathcal{K},\ t\in\mathcal{T} \tag{2}$$

$$\sum_{\substack{f\ :\ P(f,i)=j,\ P(f,i+1)=j',\\ 1<i<N_f,\ l\in\mathcal{L}_j^f,\ l'\in\mathcal{L}_{j'}^f}} (x^f_{j,l}(t) - x^f_{j',l'}(t)) \leqslant S_j(t) \qquad \forall\ j\in\mathcal{J},\ t\in\mathcal{T} \tag{3}$$

The first three sets of constraints bound the solution to respect the capacity of airports and sectors. We will therefore refer to them as capacity constraints. Constraints (1) ensure that the number of flights that may take off from an airport k at any time t will not exceed the departure capacity of airport k at time t, $D_k(t)$. Likewise, constraints (2) ensure that the number of flights that may arrive at an airport k at any time t will not exceed the departure capacity of airport k at time t, $A_k(t)$. Finally, constraints (3) ensure that for each sector j the total number of flights that may feasibly cross it will not exceed the capacity of j at any time t, $S_j(t)$. This number is expressed by the left-hand side of (3), since it counts the number of flights that has entered sector j (at any feasible level) by time t and has not yet entered in the successive sector (at any feasible level).

5.7.2 Relations among variables

The constraints (4) define the $y^f_{j,l}(t)$ variables:

$$y^f_{j,l}(t) \;\geqslant\; x^f_{j,l}(t) \;-\sum_{l' \in \mathcal{L}^f_{j'}} x^f_{j',l'}(t) \qquad \begin{cases} \forall\ f \in \mathcal{F},\ 1 < i < Nf \\ j = P(f,i),\ j' = P(f,i+1), \\ l \in \mathcal{L}^f_j,\ t \in [\underline{T}^f_j, \overline{T}^f_j + I^f_j] \end{cases} \tag{4}$$

Instead of using the max operator, we will actually define $y^f_{j,l}(t)$ through the following relation:

This definition ensures us that whenever flight f is in sector j at level l at time t then $y^f_{j,l}(t) = 1$. However, if this is not the case, then $y^f_{j,l}(t)$ is free to take either 0 or 1 value, in particular it is not bounded to take 0 value, as in the definition given at the beginning of the section. This does not actually bother us, since the first information is the only one we need.

5.7.3 Space-time connectivity constraints

$$\sum_{l \in \mathcal{L}_j^f} x_{j,l}^f(t) \leqslant 1 \qquad \forall\ f \in \mathcal{F},\ j \in P_f,\ t \in T_j^f \tag{5}$$

Constraints (5) state that each flight cannot fly inside a sector at more than one level simultaneously. (The level variation between sectors is quantized).

$$\sum_{l \in \mathcal{L}_j^f} x_{j,l}^f(t) - \sum_{l' \in \mathcal{L}_{j'}^f} x_{j',l'}^f(t + I_j^f) = 0 \qquad \begin{cases} \forall f \in \mathcal{F},\ t \in T_j^f, \\ j = P(f,i),\ j' = P(f,i+1), \\ i < N_f - 1 \end{cases} \tag{6a}$$

Constraints (6a) express sector connectivity: they state that before every flight passes from a sector j to the next one j', exactly (since we assume that the aircraft cannot change its cruise speed) I_j^f units of time must have been spent between the time t in which the flight has entered sector j and the time it enters the next one.

$$x_{j,l}^f(t) \ - \sum_{l' \in \mathcal{L}_{j'}^f \cap [l - \delta_j^f, l + \delta_j^f]} x_{j',l'}^f(t + I_j^f) \leqslant 0 \qquad \begin{cases} \forall f \in \mathcal{F},\ t \in T_j^f,\ l \in \mathcal{L}_j^f, \\ j = P(f,i),\ j' = P(f,i+1), \\ i < N_f - 1 \end{cases} \tag{6b}$$

Constraints (6b) express level connectivity: they state that if a flight f has entered sector j at level l by time t, then after I_j^f units of time it must have entered the next sector j' at a feasible level l' reachable from level l without exceeding the maximum level variation δ_j^f between j and j'.

$$\sum_{l \in \mathcal{L}_j^f \cap [l^0, l^0 + \delta_j^f]} x_{j,l}^f(t) - \ x_{k,l^0}^f(t + I_j^f) \geqslant 0 \qquad \begin{cases} \forall f \in \mathcal{F},\ t \in T_j^f, \\ j = P(f, N_f - 1), \\ k = P(f, N_f) \end{cases} \tag{6c}$$

Constraints (6c) state two requirements all together, specifying the two kind of constraints above described for the case in which the next element in a flight path is the

arrival airport. The first one is that before each flight lands at the arrival airport k, at least (in this case possibly not exactly, since we assume that the aircraft can hold on airborne on the arrival airport) I_j^f units of time must have been spent between the time t in which the flight has entered sector j to which the arrival airport belongs and the time it lands. The second request is that each flight cannot enter sector j at a level higher than $l^0 + \delta_j^f$ in order to be able to land without violating the maximum level variation.

5.7.4 Separation constraints

Given that often nominal sector capacities are underestimated compared to the actual number of flights allowed in current operations, we propose to relax sector capacity constraints, using observed capacities as right-hand side, in order to have more flexibility. Of course, we still need to maintain some safety conditions. Hence we want to impose some extra separation constraints. These constraints will be of two types. Before we proceed with the description of the constraints, we need to introduce the following set:

Definition *We define the set of airspace directed edges as:*

$$\mathcal{A} = \{a = (j, j') \mid j, j' \in \mathcal{J},\ \exists f \in \mathcal{F} : j = P(f, i),\ j' = P(f, i+1),\ 1 < i < N_f - 1\}$$

With abuse of notation we will write $a = (j, j') \in P_f$ *if* $(j, j') \in \mathcal{A}$ *and* $j, j' \in P_f$.

This shows that our problem input data implicitly define a directed graph G, whose nodes are airports and sectors and whose edges are those just defined. ($G = (\mathcal{K} \cup \mathcal{J}, \mathcal{A})$). We shall call this graph the *airspace flow graph*. The first type of constraints involves flights that are in a "potential chasing conflict" (CC). By that we mean that their paths share a sector j and that the flights are entering the sector with concordant directions, more precisely if they are entering using concordant edges. Whether or not the edges are concordant is an information carried by the sets K^C defined below, which are provided as input data. Intuitively, one could think of

concordant edges as edges with the same head and with an in-between angle going from 0° to 90°. As an example of such a situation see **Figure 5.4**: flight $f1$ is in a potential chasing conflict with flights $f2$, $f3$ and $f7$. In this case, either we impose a large enough vertical separation (at least l^C_{sep} flight levels), or, if the vertical separation is under this threshold, we impose a large enough horizontal separation (the flights can enter the same sector only if they are separated by a large enough time distance t^C_{sep}).

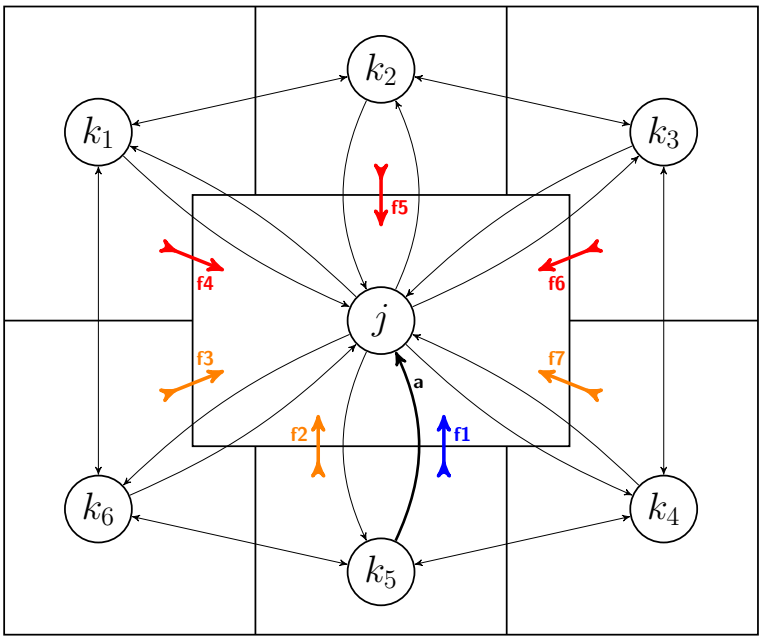


Figure 5.4: a depiction of the two possible conflict types: (CC) and (OC). We are considering the situation from the point of view of flight **f1**, which is travelling through the edge $a = (k_5, j)$. We have:

$$K^C(a) = \{k_4, k_5, k_6\} \qquad K^O(a) = \{k_1, k_2, k_3\}$$

The second type of constraints involves flights that are in an "potential opposing conflict" (OC). In this case we mean that their paths share a sector j and that the

flights are entering sector j using opposite edges. As in the previous case whether or not two edges are opposite is an information given by the sets K^O. $K^O(k,j)$ contains all the edges with head in j that are not in $K^C(k,j)$. Hence, these sets come from input data. Again, a naive way to think of opposite edges is as edges with same head and with an in-between angle going from 90° and 180°. As an example of this situation see **Figure 5.4**: flight $f1$ is in a potential opposing conflict with flights $f4$, $f5$ and $f6$. In this case we want to impose a very strong vertical separation, meaning that they can be in the same sector only one at a time, unless they are maintaining a safe vertical separation l^O_{sep} (normally, $l^O_{sep} > l^C_{sep}$). That is: as long as one flight is in a sector no other flight can enter the same sector if it is not at a sufficiently higher or lower level. This seems a reasonably strong request to make and it should guarantee sufficient safety conditions. However, if we don't impose any other condition, as soon as one flight crossing a sector j has left it, another one can enter in j at the same or similar flight level. In order to implement a flexible and safe separation policy, it's worth to impose a (small) horizontal separation too. In this way if a flight has just left a sector j, any other flight coming from an opposite direction (edge) need to wait some time, namely t^O_{sep}, before entering j. We now give the following

Definitions

1. For $a \in \mathcal{A},\ a = (j,j')$, we denote the set of potential chasing conflicting sectors centered in a with:

$$K^C(a) = K^C(j,j') = \{k \in \mathcal{J} \mid (k,j') \in \mathcal{A},\ (k,j') \textit{ is in a potential chasing conflict with } (j,j')\}$$

Note that j should always be in $K^C(j,j')$.

2. For $a \in \mathcal{A},\ a = (j,j')$, we denote the set of potential opposing conflicting sectors opposite to a with:

$$\begin{aligned} K^O(a) = K^O(j,j') &= \{k \in \mathcal{J} \mid (k,j') \in \mathcal{A},\ (k,j') \textit{ is in a potential opposing conflict with } (j,j')\} \\ &= \{k \in \ \mathcal{J} \mid\ (k,j') \in \mathcal{A}\} \setminus K^C(j'j') \end{aligned}$$

The potential conflict edge sets are such that for all $a = (j, j') \in \mathcal{A}$ hold the relations $K^C(a) \cap K^O(a) = 0$ and $K^C(a) \cup K^O(a) = \{(k, j') \mid (k, j') \in \mathcal{A}\}$. The first relation states that no edge can be simultaneously in a potential chasing conflict and in a potential opposing conflict with another edge, while the second relation states that given an edge $a = (j, j')$, every edge entering sector j' must be either in a potential chasing conflict or in a potential opposing conflict with a. Furthermore, for the sake of consistency, the potential conflict edges sets should be built in such a way that the following symmetric property holds: $k \in K^C(j, j') \Rightarrow j \in K^C(k, j')$ and $k \in K^O(j, j') \Rightarrow j \in K^O(k, j')$. These properties state that if an edge (k, j') is in a potential chasing/opposing conflict with another edge (j, j') then (j, j') is in a potential chasing/opposing conflict with (k, j')

Equipped with the above definitions we are now ready to give a formal description of the constraints previously introduced.

$$\sum_{s=t}^{t+t^C_{sep}} \sum_{m=l}^{l+l^C_{sep}} \sum_{k \in K^C(a)} \sum_{\substack{f\,:\,(k,j') \in P_f \\ m \in \mathcal{L}^f_{j'},\ s \in T^f_{j'}}} \Big(x^f_{j',m}(s) - x^f_{j',m}(s-1) \Big) \leqslant 1 \qquad \begin{cases} \forall\ a = (j, j') \in \mathcal{A}, \\ l \in \mathcal{L},\ t \in \mathcal{T} \end{cases} \tag{7a}$$

$$y^f_{j',l}(t) \ + \ u_{a,l}(t) \leqslant 1 \qquad \begin{cases} \forall\ a = (j, j') \in \mathcal{A}, \\ f \in \mathcal{F} \text{ s.t. } a \in P_f \\ l \in \mathcal{L}^f_{j'},\ t \in [\underline{T}^f_{j'}, \overline{T}^f_{j'} + I^f_{j'}] \end{cases} \tag{7b}$$

$$\sum_{s=t}^{t+t^O_{sep}} \sum_{m=l-l^O_{sep}}^{l+l^O_{sep}} \sum_{k \in K^O(a)} \sum_{\substack{f'\,:\,(k,j') \in P_{f'} \\ m \in \mathcal{L}^{f'}_{j'},\ s \in T^{f'}_{j'}}} \Big(x^{f'}_{j',m}(s) - x^{f'}_{j',m}(s-1) \Big) \leqslant M_a u_{a,l}(t) \qquad \begin{cases} \forall\ a = (j, j') \in \mathcal{A}, \\ l \in \mathcal{L},\ t \in \mathcal{T} \end{cases} \tag{7c}$$

Constraints (7a) are imposing conditions on the entrance time (remember that the variables difference $x^f_{j,l}(t) - x^f_{j,l}(t-1)$ takes value 1 if and only if flight f enters sector j at level l at time t). They are defining a 3-dimensional (time,level,edge) conflict zone in which no more than a flight is allowed to enter a certain sector.

Constraints (7b) and (7c) set together stronger requirements. To impose the conditions discussed above, we need to have information not just on the entrance time of a flight in a sector but on the entire time period spent by a flight in a sector. That is why we make use of $y^f_{j,l}(t)$ variables (remember that $y^f_{j,l}(t)$ takes value 1 whenever flight f is crossing sector j at level l at time t): in case of a potential opposing conflict we want to control flight distances not just at the entrance time but during all the permanence of a flight in the sector. These constraints then impose that if a flight f is (still) crossing sector j' at level l at time t coming from sector j, no flight coming from an opposite sector k can cross the same sector j' unless it cross it either maintaining a vertical distance larger than l^O_{sep} or waiting some time (at least t^O_{sep}) after the moment in which f leaves the sector. In this way we are guaranteeing that flights that are in a potential opposing conflict can't cross simultaneously the same sector unless they are vertically well separated. Let us now explain how the two constraints sets work together to translate this requirement: for every edge $a \in A$ we consider in (7b) a flight f that uses that edge to enter a sector j' coming from another sector j. Constraints (7b) tell us that whenever f is travelling inside sector j' at a certain (feasible) time t and at a certain (feasible) level (so that $y^f_{j',l}(t) = 1$) the binary variable u, depending on the same edge, same level and same time, is forced to take value 0. Here is where constraints (7c) come into play: they state that if $u_{a,l}(t) = 0$ no flight can enter sector j' using an edge in potential opposing conflict with a, unless they enter after some time t^O_{sep} or maintaining a sufficiently high vertical separation (at least l^O_{sep} levels). If otherwise $y^f_{j',l}(t) = 0$, meaning that flight f is not crossing sector j' at time t at level l, we don't want to make any imposition on the flights in a potential opposing conflict with f. In fact, in this case constraint (7b) is not imposing any restriction on $u_{a,l}(t)$, which is therefore free to take either 0 or 1 value. This allow the LHS of (7c) to take any value between 0 and M_a. This latter is a constant counting the number of flights possibly using an edge in a potential opposing conflict with a. This constant can be further refined defining it as the number of flights that possibly use an edge in a potential opposing conflict with edge a at a time $s \in [t, t + t^O_{sep}]$, at a level $m \in [l - l^O_{sep}, l + l^O_{sep}]$. This means that, as desired, no imposition is actually made on the number of flights coming from an opposite direction to that of the edge a.

One way in which safety can be improved is restricting the set K^C and consequently enlarging K^O. Another one is to increase vertical and horizontal separation in both constraints set, that is to take big $t^C_{sep},\ l^C_{sep},\ t^O_{sep}$ and l^O_{sep}. Furthermore, for the sake of flexibility the parameters of flights separation may be sector dependent. In other words we should write $t^C_{sep}(j),\ l^C_{sep}(j),\ t^O_{sep}(j)$ and $l^O_{sep}(j)$.

Observation

Instead of inequalities (7*b*) *and (7c) one can formally impose opposing separation constraints through the following relations:*

$$y^f_{j',l}(t) + \sum_{\substack{s=t \\ s\in T^{f'}_{j'}}}^{t+t^O_{sep}} \sum_{\substack{m=l-l^O_{sep} \\ m\in\mathcal{L}^{f'}_{j'}}}^{l+l^O_{sep}} \left(x^{f'}_{j',m}(s) - x^{f'}_{j',m}(s-1)\right) \leqslant 1 \qquad \begin{cases} \forall\ f\in\mathcal{F},\ a=(j,j')\in P_f, \\ k\in K^O(a),\ f'\ :\ (k,j')\in P_{f'}, \\ l\in\mathcal{L}^f_{j'},\ t\in[\underline{T}^f_{j'}, \overline{T}^f_{j'} + I^f_{j'}] \end{cases} \qquad (7b)'$$

The idea here is to define a constraint for each couple of flights that are in a possible opposing conflict. This is of course a possibility, but the number of opposing separation constraints would dramatically increase. In fact the number of opposing separation constraints is in the two cases:

$$(7b),\ (7c):\ |\mathcal{A}||\mathcal{L}||\mathcal{T}| + \sum_{(j,j')\in\mathcal{A}} \sum_{\substack{f \text{ s.t.} \\ (j,j')\in P_f}} |\mathcal{L}^f_{j'}|(|T^f_{j'}| + I^f_{j'})$$

$$(7b)':\ \sum_{f\in\mathcal{F}} \sum_{(j,j')\in\mathcal{A}} \sum_{k\in K^O(j,j')} \sum_{f:\ (k,j')\in P_{f'}} |\mathcal{L}^f_{j'}|(|T^f_{j'}| + I^f_{j'})$$

An upper bound for the number of (7*b*), (7*c*) constraints is $|\mathcal{A}|(|\mathcal{L}||\mathcal{T}| + WY(D+Z))$, an upper bound for the number of $(7b)'$ constraints is $|\mathcal{F}||\mathcal{A}|KWY(D+Z)$, where $W = \max_{a\in\mathcal{A}} |\{f \text{ s.t. } a\in P_f\}|$ and $K = \max_{a\in\mathcal{A}} |K^O(a)|$.

5.7.5 Variables shape and domains

$$x^f_{k,0}(\overline{T}^f_k) = 1 \qquad \forall\ f \in \mathcal{F},\ k = P(f, N_f) \tag{8}$$

$$x^f_{j,l}(t) - x^f_{j,l}(t-1) \geqslant 0 \qquad \forall\ f \in \mathcal{F},\ j \in P_f,\ l \in \mathcal{L}^f_j,\ t \in T^f_j \tag{9}$$

$$x^f_{j,l}(t),\ y^f_{j,l}(s),\ u_{a,m}(r) \in \{0,1\} \qquad \begin{cases} \forall\ f \in \mathcal{F}\ j \in P_f,\ l \in \mathcal{L}^f_j, \\ t \in T^f_j,\ s \in [\underline{T}^f_j, \overline{T}^f_j + I^f_j] \\ a \in \mathcal{A},\ m \in \mathcal{L},\ r \in \mathcal{T} \end{cases} \tag{10}$$

Constraints (8) ensure that each flight must land at the arrival airport by the last feasible time $\overline{T}^f_k$. This is crucial in order to exclude the trivial solution $x^f_{j,l}(t) = 0\ \ \forall f \in \mathcal{F},\ j \in \mathcal{J},\ l \in \mathcal{L},\ t \in \mathcal{T}$

Constraints (9) assure the time consistency of $x^f_{j,l}(t)$ variables: if flight f has entered sector j at level l by time $t-1$ then it must has entered sector j at level l by time t (and so by every other time $s \geqslant t$).

Constraints (10) set the model variables to be binary.

5.8 Simple trajectories

The proposed model includes constraints that impose that a flight's level variation between two consecutive sectors doesn't exceed a certain fixed maximal variation. One more issue that one may want to deal with is how these level variations affect the trajectory of a flight. In fact, one would like to have flights with a simple trajectory.

By a simple trajectory we mean a trajectory which is divided into exactly two simple flight phases, a climb phase and descent phase. This is how an usual flight trajectory is divided. Our model, however, doesn't exclude that unpleasant zigzagging trajectories are assigned to a flight. As an example see **Figure 5.5**: the flight depicted has a trajectory which is not simple. It is easy to see that such a situation can only arise when the number of sectors crossed by a fight is larger than 5. In order to approach this problem we define the following two type of binary variables, which we will call respectively climbing and descending variables:

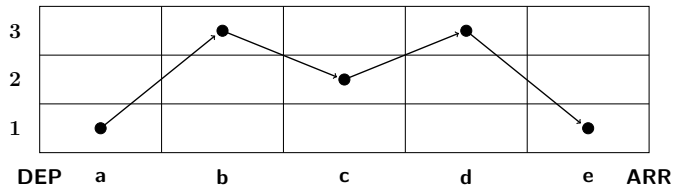


Figure 5.5: A zigzagging trajectory

$$\widehat{z}^f_{j,l}(t) = \begin{cases} 1 & \text{if flight } f \text{ has changed its flight level between sector } j \text{ and sector } succ(j) \\ & \text{from level } l \text{ to a (strictly) } \textbf{higher} \text{ one} \\ 0 & \text{otherwise} \end{cases}$$

$$\breve{z}^f_{j,l}(t) = \begin{cases} 1 & \text{if flight } f \text{ has changed its flight level between sector } j \text{ and sector } succ(j) \\ & \text{from level } l \text{ to a (strictly) } \textbf{lower} \text{ one} \\ 0 & \text{otherwise} \end{cases}$$

Constraints (11a) and (11b) include the definition of climbing and descending variables into the model.

$$\widehat{z}^f_{j,l}(t) \;\geqslant\; (x^f_{j,l}(t) - x^f_{j,l}(t-1)) \;\; - \sum_{\substack{j'=succ(j),\\ m\in\mathcal{L}^f_{j'},\; m\leqslant\, l}} (x^f_{j',m}(t+I^f_j) - x^f_{j',m}(t+I^f_j-1) \tag{11a}$$

$$\breve{z}^f_{j,l}(t) \;\geqslant\; (x^f_{j,l}(t) - x^f_{j,l}(t-1)) \;\; - \sum_{\substack{j'=succ(j),\\ m\in\mathcal{L}^f_{j'},\; m\geqslant\, l}} (x^f_{j',m}(t+I^f_j) - x^f_{j',m}(t+I^f_j-1) \tag{11b}$$

$$\forall\; f\in\mathcal{F},\; \forall\; j\in\; P_f\backslash\mathcal{K},\; l\in\mathcal{L}^f_j,\; t\in T^f_j\;:$$

Actually, if j is the first sector crossed by a flight right after take off we don't need neither $\widehat{z}^f_{j,l}(t)$ nor $\breve{z}^f_{j,l}(t)$ variables. That's because a flight is always forced to climb during take off. For the analogous reason (a flight is always forced to descend during the landing) we do not need neither $\widehat{z}^f_{j,l}(t)$ nor $\breve{z}^f_{j,l}(t)$ variables if j is the last sector crossed by a flight right before landing.

We are now ready to impose the simple trajectory constraints.

$$\sum_{l\in\mathcal{L}^f_j} \breve{z}^f_{j,l}(t) \; + \sum_{m\in\mathcal{L}^f_{j'}} \widehat{z}^f_{k,m}(t+\sum_{h=i}^{n-1} I^f_h) \leqslant 1 \qquad \begin{cases} \forall\ f\in\mathcal{F} \text{ s.t. } N_f \geqslant 7, \\ j = P(f,i),\ 1<i<N_f-2, \\ k = P(f,n),\ i<n<N_f-1,\ t\in\mathcal{T}^f_j \end{cases} \tag{12}$$

Constraints (12) ensure that whenever a flight has descent between a sector $j = P(f,i)$ and its successor $j' = P(f,i+1)$ from level l to a lower one, it can't climb from $k = P(f,n)$ to $k' = P(f,n+1)$ from a certain level m to a higher one, whenever k is a sector which follows sector j in the path of f. In other words it guarantees that once an aircraft has started descent it can never climb again to an upper level, hence ensuring simple trajectories.

5.9 Two classes of valid inequalities

In the following we propose two classes of valid inequalities for the set of feasible solutions of ATFMP. Their purpose is to strengthen the polyhedral structure of the formulation. This inequalities are inspired by the valid inequalities presented by Bertsimas, Lulli & Odoni in [4]. In fact, the object of that paper is rerouting. The authors propose two classes of valid inequalities, exploiting the information of actual feasible routes in order to make some deductions, then formalized into these inequalities. We replicate their arguments in the scope of level distribution, which can be seen as a sort of vertical rerouting. For a comfortable notation we give the following

Definitions

1. *Given* $f \in \mathcal{F},\ (j, j') \in P_f, l \in \mathcal{L}_j^f$ *we define the set of successor levels of level* l *in sector* j *for flight* f *as:*

$$\mathcal{S}_j^f(l) := \mathcal{L}_{j'}^f \cap [l - \delta_j^f, l + \delta_j^f]$$

2. *Given* $f \in \mathcal{F},\ (j, j') \in P_f, l' \in \mathcal{L}_j^f$ *we define the set of predecessor levels of level* l' *in sector* j' *for flight* f *as:*

$$\mathcal{P}_{j'}^f(l') := \{m \in \mathcal{L}_j^f \mid l' \in [m - \delta_j^f, m + \delta_j^f]\ \}$$

The first set contains all the levels in sector j' feasibly reachable by flight f from level l in sector j. Similarly, the second set contains all the levels in sector j from which flight f can feasibly reach level l' in sector j'. We are now ready to present the following two symmetrical proposition:

Proposition 1 (Fork level inequalities)
The constraints

$$x_{j,l}^f(t) \ \geqslant \sum_{\substack{l' \in S_j^f(l)\ s.t. \\ |\mathcal{P}_{j'}^f(l')|=1}} x_{j',l'}^f(t + I_j^f) \qquad \begin{cases} \forall f \in \mathcal{F},\ (j, j') \in P_f \\ l \in \mathcal{L}_j^f,\ t \in T_j^f \end{cases}$$

are valid inequalities for the set of feasible solutions of ATFMP.

Proof. The inequalities of Proposition 1 state that if a flight f has not entered sector j at level l by time t ($x_{j,l}^f(t) = 0$) it will not enter sector j' by time $t + I_j^f$ at any of the levels feasibly reachable from l, unless these level can be feasibly reached from other levels belonging to j. A formal proof is the following: fix $f \in \mathcal{F},\ (j, j') \in P_f$

$l \in \mathcal{L}_j^f$, $t \in T_j^f$ and suppose that $\sum_{\substack{l' \in S_j^f(l) \text{ s.t.} \\ |\mathcal{P}_{j'}^f(l')|=1}} x_{j',l'}^f(t + I_j^f) = 1$. We need to prove that $x_{j,l}^f(t) = 1$. By constraints (5) we deduce that $\sum_{l' \in \mathcal{L}_{j'}^f} x_{j',l'}^f(t + I_j^f) = 1$. Hence by sector connectivity constraints (6a) we have $\sum_{m \in \mathcal{L}_j^f} x_{j,m}^f(t) = 1$. If $x_{j,m}^f(t) = 1$ with $m \neq l$ then for constraints (6b) $\sum_{l' \in \mathcal{S}_j^f(m)} x_{j',l'}^f(t + I_j^f) = 1$ must hold. Then, since $\mathcal{S}_j^f(m) \cap \{l' \in \mathcal{S}_j^f(l) \text{ s.t. } |\mathcal{P}_{j'}^f(l')| = 1\} = \varnothing$, for constraints (5) $x_{j',l'}^f(t + I_j^f) = 0$ $\forall\ l' \in \mathcal{S}_j^f(l)$ s.t $|\mathcal{P}_{j'}^f(l')| = 1$, which contradicts our hypothesis. Hence $x_{j,l}^f(t) = 1$. □

For example, see **Figure 5.6**. If flight f has not entered sector j at level 5 by time t ($x_{j,5}^f(t) = 0$), it will not enter sector j' by time $t + I_j^f$ neither at level 4 nor at level 5, since they can be feasibly reached only from level 5. Hence $x_{j',4}^f(t + I_j^f) + x_{j',5}^f(t + I_j^f) = 0$. However, since level 6 is feasibly reachable from level 7 of j, $x_{j,5}^f(t) = 0$ does not imply that $x_{j',6}^f(t + I_j^f) = 0$.

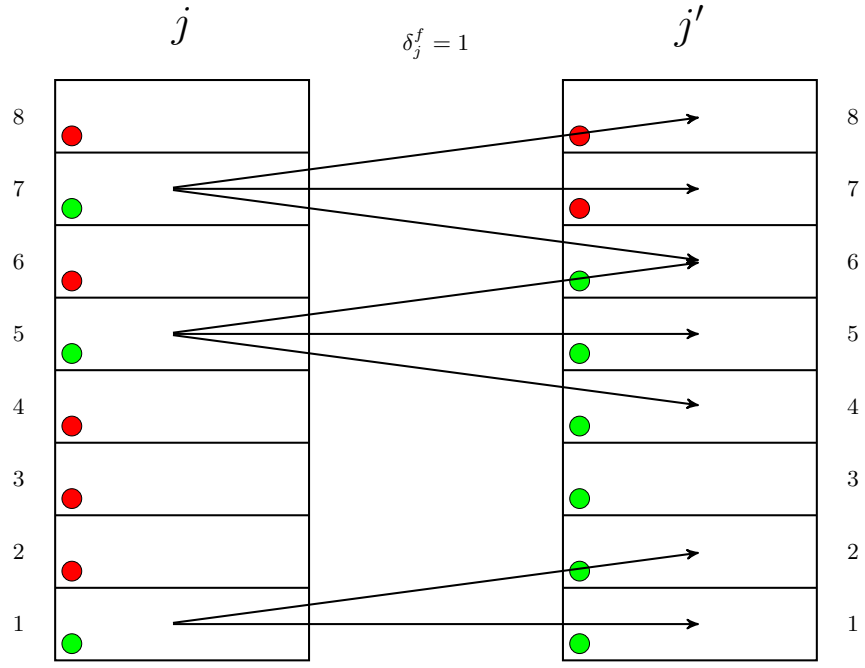


Figure 5.6: A representation of fork&joint inequalities: it depicts a situation in which a flight f passes from a sector j to the subsequent sector j' with a maximum level variation δ_j^f set to 1. The red and green circles indicate which are the feasible levels for f.

Proposition 2 (Joint level inequalities)
The constraints

$$x^f_{j',l'}(t+I^f_j) \geqslant \sum_{\substack{l\in\mathcal{P}^f_{j'}(l') \text{ s.t.} \\ |S^f_j(l)|=1}} x^f_{j,l}(t) \qquad \begin{cases} \forall f\in\mathcal{F},\ (j,j')\in P_f \\ l'\in\mathcal{L}^f_{j'},\ t\in T^f_j \end{cases}$$

are valid inequalities for the set of feasible solutions of ATFMP.

Proof. A symmetric argument for the level joint inequalities of Proposition 2 can be made: they state that if a flight f has not entered sector j' at level l' by time $t+I^f_j$ ($x^f_{j',l'}(t+I^f_j)=0$) it cannot has entered sector j by time t at any of the levels from which l' can be feasibly reachable, unless these level can feasibly reach other levels belonging to j'. The symmetrical formal proof is the following: fix $f\in\mathcal{F},\ (j,j')\in P_f\ l'\in\mathcal{L}^f_{j'},\ t\in T^f_j$ and suppose that $\sum_{\substack{l\in P^f_{j'}(l') \text{ s.t.} \\ |\mathcal{S}^f_j(l)|=1}} x^f_{j,l}(t)=1$. We need to prove that $x^f_{j',l'}(t+I^f_j)=1$. By constraints (5) we deduce that $\sum_{l\in\ \mathcal{L}^f_j} x^f_{j,l}(t)=1$. Hence by sector connectivity constraints (6a) we have $\sum_{m\in\mathcal{L}^f_{j'}} x^f_{j',m}(t+I^f_j)=1$. If $x^f_{j',m}(t+I^f_j)=1$ with $m\neq l'$ then $\sum_{l\in\mathcal{P}^f_{j'}(m)} x^f_{j,l}(t)=1$ must hold. Otherwise, in fact, if $\sum_{l\in\mathcal{P}^f_{j'}(m)} x^f_{j,l}(t)=0$ for constraints (6b) $x^f_{j',m}(t+I^f_j)=0$. Then, since $\mathcal{P}^f_{j'}(m)\cap\{l\in\mathcal{S}^f_{j'}(l') \text{ s.t. } |\mathcal{S}^f_j(l)|=1\}=\varnothing$, for constraints (5) $x^f_{j,l}(t)=0\ \forall\ l\in\mathcal{P}^f_{j'}(l')$ s.t $|\mathcal{S}^f_j(l)|=1$, which contradicts our hypothesis. Hence $x^f_{j',l'}(t+I^f_j)=1$. □

For example, see again **Figure 5.6**. If flight f has not entered sector j' at level 6 by time $t+I^f_j$ ($x^f_{j,6}(t+I^f_j)=0$), it cannot has entered sector j by time t at level 7, since it can feasibly reach only level 6. Hence $x^f_{j',7}(t)=0$. However, since level 5 can feasibly reach levels 4 and 5 of j', $x^f_{j,6}(t+I^f_j)=0$ does not imply that $x^f_{j',5}(t)=0$.

Observation

Fork level inequalities looks very similar to level connectivity constraints (6b) of our model:

$$x^f_{j,l}(t) \leqslant \sum_{l' \in \mathcal{S}^f_j(l)} x^f_{j',l'}(t + I^f_j) \qquad \begin{cases} \forall f \in \mathcal{F},\ (j,j') \in P_f \\ l \in \mathcal{L}^f_j,\ t \in T^f_j \end{cases} \tag{6b}$$

One should not be deceived: the opposite sign of the inequality, along with the fact the the right-hand side summation is made over levels with exactly one predecessor, implies that the inequalities exposed above carry an information that is contained in the model only indirectly. In fact, level connectivity constraints state the is a flight has reached a certain sector j at a certain level l by a certain time t, it then have to reach the subsequent sector j' by the time $t + I^j_f$ at a feasible level $l' \in \mathcal{S}^f_j(l)$. If it has not reached sector j at level l at time t no implication is available for variable $x^f_{j',l'}(t + I^f_j)$. As showed in the proof of **Proposition 1**, in order to deduce fork level inequalities we need to step through some not trivial proof passages involving other constraints types. It is then worth to explicit this information.

5.10 Size of the formulation

Let D be the maximum cardinality of the set of feasible times for flight f to be in sector j taken over all f and j, i.e.

$$D = \max_{f \in \mathcal{F}, j \in P_f} |T^f_j|$$

Let then

$$X = \max_{f \in \mathcal{F}} |N_f|, \qquad Y = \max_{f \in \mathcal{F}, j \in P_f} |\mathcal{L}^f_j| \qquad \text{and} \qquad Z = \max_{f \in \mathcal{F}, j \in P_f} \mathcal{I}^f_j$$

be respectively the maximum number of sectors that a flight passes through along its route and the maximum number of levels at which a flight can cross a sector, taken over all flights. $X \geqslant 3$, since departure and arrival airports are always counted as sectors on a flight's path and a path always contains at least one sector.

We recall that:

- $|\mathcal{F}|$ is the total number of flights
- $|\mathcal{K}|$ is the total number of airports
- $|\mathcal{J}|$ is the total number of sectors
- $|\mathcal{L}|$ is the total number of levels
- $|\mathcal{T}|$ is the total number of time periods
- $|A|$ is the total number of edges

The actual number of variables $x^f_{j,l}(t)$ is

$$\sum_{f\in\mathcal{F}}\sum_{j\in P_f}|\mathcal{L}^f_j||T^f_j|$$

An upper bound to the number of variables $x^f_{j,l}(t)$ is then $|\mathcal{F}|XYD$.
The actual number of variables $y^f_{j,l}(t)$ is

$$\sum_{f\in\mathcal{F}}\sum_{j\in P_f}|\mathcal{L}^f_j|(|T^f_j|+I^f_j)$$

An upper bound to the number of variables $y^f_{j,l}(t)$ is then $|\mathcal{F}|XY(D+Z)$.
The actual number of variables $u_{a,l}(t)$ is

$$|\mathcal{A}||\mathcal{L}||\mathcal{T}|$$

The exact number of constraints is:

$$2|\mathcal{K}||\mathcal{T}| + |\mathcal{J}||\mathcal{T}| + \sum_{f\in\mathcal{F}} \sum_{j=P(f,2)}^{P(f,N_f-1)} |\mathcal{L}_j^f|(|T_j^f| + I_j^f) \qquad (1),(2),(3),(4)$$

$$+ \sum_{f\in\mathcal{F}} \sum_{j\in P_f} |\mathcal{L}_j^f||T_j^f| + \sum_{f\in\mathcal{F}} \sum_{j=P(f,1)}^{P(f,N_f-2)} (1 + |\mathcal{L}_j^f|)|T_j^f| \qquad (5),(6a),(6b)$$

$$+ \sum_{f\in\mathcal{F}} |T^f_{P(f,N_f-1)}| + \sum_{(j,j')\in\mathcal{A}} \sum_{\substack{f \text{ s.t.} \\ (j,j')\in P_f}} |\mathcal{L}_{j'}^f|(|T_{j'}^f| + I_{j'}^f) \qquad (6c),(7b)$$

$$+ 3|\mathcal{A}||\mathcal{L}||\mathcal{T}| + |\mathcal{F}| + \sum_{f\in\mathcal{F}} \sum_{j\in P_f} |\mathcal{L}_j^f|(3|T_j^f| + I_j^f) \qquad (7a),(7c),(8),(9),(10)$$

An upper bound on the number of constraints is:

$$\begin{aligned} &2|\mathcal{K}||\mathcal{T}| + |\mathcal{J}||\mathcal{T}| + |\mathcal{F}|XY(D+Z) \\ &+|\mathcal{F}|XYD + |\mathcal{F}|X(1+Y)D \\ &+|\mathcal{F}|D + |\mathcal{A}||\mathcal{F}|Y(D+Z) \\ &+3|\mathcal{A}||\mathcal{L}||\mathcal{T}| + |\mathcal{F}| + |\mathcal{F}|XY(3D+Z) \end{aligned}$$

which can be rewritten as

$$\begin{aligned} &2|\mathcal{K}||\mathcal{T}| + |\mathcal{J}||\mathcal{T}| + 3|\mathcal{A}||\mathcal{L}||\mathcal{T}| + |\mathcal{F}|XY(6D+2Z) \\ &+ |\mathcal{F}|D(X+1) + |\mathcal{A}||\mathcal{F}|Y(D+Z) + |\mathcal{F}| \end{aligned}$$

Let us make some computations to see the model size in two different instances. We first consider a realistic instance with 3000 flights, 112 sectors, 20 airports, 10 levels, 45 times and 5×112 edges (supposing that on average from each sector other 5 sector are reachable). Suppose then that the upper bounds D, X, Y and Z are 4, 5, 10 and 3 respectively.

The upper bounds for the number of variables are:

$$\begin{aligned} x^f_{j,l}(t): &\quad 3000 \times 5 \times 10 \times 4 = 600\,000 \\ y^f_{j,l}(t): &\quad 3000 \times 5 \times 10 \times (4+3) = 1\,050\,000 \\ u_{a,l}(t): &\quad 5 \times 12 \times 10 \times 45 = 252\,000 \end{aligned}$$

An upper bound for the total number of variables is then 1 902 000, almost 2 million variables. An upper bound for the number of constraints is 1 122 937 840, more than one billion constraints. These number are really big and although such a problem is still treatable, there's no hope to have it solved in less then hours.

As **Chapter 7** will testify, for smaller instances the model dimensions are much more treatable. For example, let us take the "super hexagon" airspace of **Section 7.1**. We have 75 flights, 13 sectors, 6 airports, 9 levels, 45 times and 45 edges. The upper bounds D, X, Y and Z are 10, 7, 9 and 4 respectively. The upper bounds for the number of variables in this case are:

$$\begin{aligned} x^f_{j,l}(t): &\quad 75 \times 7 \times 9 \times 10 = 47\,250 \\ y^f_{j,l}(t): &\quad 75 \times 7 \times 9 \times (10+4) = 66\,150 \\ u_{a,l}(t): &\quad 45 \times 9 \times 45 = 18\,225 \end{aligned}$$

An upper bound for the total number of variables is then 131 625. An upper bound for the number of constraints is 808 425. These numbers are far more treatable.

Chapter 6

Model implementation

For the implementation of our model, we have used IBM ILOG CPLEX Optimization Studio (version 12.6.0) using its Python (version 2.7) interface as the modelling language. For some insight on CPLEX and Python see **Subsection 3.2.2**. Every single problem instance can be described by the data file format described in The input data are those corresponding to to the notation defined in **Section 5.3**. This data file is then readable by the Python code through which we interface with the CPLEX solver. This program then writes the specific model arising from these data. In particular it defines the objective function and the constraints for that particular instance. The way in which Python defines the objective function and the inequalities is described in **Section 6.2**. Finally, in the **Appendix** are entirely reported a data file sample, named `data_super_hexagon_15_NO_SEP.txt`, which describes the 15 flight instance with no separation constrains and nominal capacities on the "super hexagon" airspace of **Section 7.1**, and the complete Python program that implements our model, named `ATFMP.py`.

6.1 Data file format description

In this section we describe the data file format that we used to describe a specific instance of the ATFMP. This is a `.txt` file containing all the problem input data as listed in **Section 5.3**, written in Python. This file is to be read by the code which models the problem instance and it then passes the to the CPLEX solver. Let us start with the description. First of all are defined the set of flights $\mathcal{F}$, the set of airports $\mathcal{K}$, the set of sectors $\mathcal{J}$, the set of levels $\mathcal{L}$ and the set of times $\mathcal{T}$. We implemented the sets as Python lists, the most common ordered iterable in Python. We defined flights, levels and times sets by using the Python feature of list comprehension.

```
## FLIGHTS
flights = ["f%d"%n for n in range(1,N_f)]

## AIRPORTS
airports = ["ABC",...,"XYZ"]

## SECTORS
sectors = ["a",...,"z",]

## LEVELS
levels = [l for l in range(N_l)]

## TIME WINDOWS
times = [t for t in range(N_t)]
```

Flights, sectors and airports are represented by strings, levels and times by integers. We then define the set of paths P_f using a dictionary: to each flight is associated a tuple (immutable iterable in Python, hence ordered) which contains the flight's path. The airports departure and arrival capacities (D_k, A_k) as well as sectors capacities S_j are defined using dictionaries.

```
## PATHS
paths = {f:(k_1,j_1,j_2,j_3,k_2),....}

## AIRPORTS CAPACITIES
airp_capacities = {ABC:(ABC_dep_cap,ABC_arr_cap),...}

## SECTORS CAPACITIES
sect_capacities = {j:j_cap,...}
```

Are then defined the crossing times: to each flight is associated a tuple whose components are the crossing times for each sector in the flight's path. Note that the first time of the tuple is 0, because the time to "cross" departure airports is 0. The feasible departure times are given assigning to each flight the first and the last feasible time for departure. To each flight is then assigned the last feasible time for arrival.

```
## SECTORS CROSSING TIMES
crossing_times = {f:(0,t_1,...),...}

## (FIRST,LAST) FEASIBLE DEPARTURE TIMES
FL_dep_times = {f:(t_first_dep,t_last_dep),...}

## LAST FEASIBLE ARRIVAL TIMES
last_arr_times = {f:t_last_arr,...}
```

Afterwards, we define the feasible levels using nested dictionaries: to each flight is assigned a dictionary which associate to every sector in the flight's path the feasible levels for that flight in that sector. The only feasible level for airports is 0. Analogously, we define the maximum level variations from a certain sector/airport in a flight's path to the next one. Then, for each edge of the airspace flow graph G is assigned the set of potential opposing conflict sectors. Note that the information of which are the edges of the graph is an implicit information carried by all flights path. Thus, given the set of potential opposing conflict sectors, the set of potential chasing conflict sectprs is implicitly given also . These two implicit information are made explicit by the code `ATFM.py` (lines 111-127).

```
## FEASIBLE LEVELS
feas_levels = {f:{k:[0],j:[l_1_j,...,l_n_j],...}

## MAXIMUM LEVEL VARIATIONS
max_level_var = {f:{k:max_lev_var_k,j:max_lev_var_j,...},...}

## OPPOSING CONFLICT SETS
opposing_conflict = {(j_1,j_2):[j_opp,...],...}
```

Finally, are naturally defined separation constants and delay costs.

```
## CHASING CONFLICT SEPARATION
chase_time_separation = cts
chase_level_separation = cls
```

```
## OPPOSING CONFLICT SEPARATION
opp_time_separation = ots
opp_level_separation = ols

## GROUND HOLDING COSTS
grh_costs = {f:grh_cost_f,...}

## AIRBORNE DELAY COSTS
abd_costs = {f:abd_cost_f,...}
```

6.2 Code description

Let us now describe the Python implementation of the model. The code that we are going to illustrate is entirely reported in the **Appendix**. The description of the first program is the following:

lines 1-3 The CPLEX module is imported and a CPLEX problem object is created.

ll. 4-8 A `.txt` data file of the form described in the previous section is imported and read

ll. 10-62 "Reading functions" are defined. They extrapolate from a variable name, which is a string of the form `"x[f,j,l,t]"`, the variable associated flight, sector, level and time

ll. 64-86 "Utility functions" are defined. They made explicit some information carried implicitly by the data file, as the departure and arrival airports, the flights minimal length, the number of sector crossed by a flight, the scheduled and the feasible arrival times

ll. 88-89 The constant M_a is defined (possibly this information could be written in the data file)

ll. 91-109 The feasible times intervals $T_j^f = [\underline{T}_j^f, \overline{T}_j^f + I_j^f]$ are defined for each flight and sector on its path

ll. 111-127 The edges of the airspace are constructed as well as the set of possible chasing conflict sectors.

ll. 129-150 Variables names, which are strings, are defined along with a variables list containing all the variables

ll. 152-164 The objective function minus the additive constant $(c_a^f - c_g^f)d_f - c_a^f r_f$ is defined

ll. 166-169 The additive constant $(c_a^f - c_g^f)d_f - c_a^f r_f$ is defined

ll. 171-188 Two classes of constraints are defined to pre-process some information: the $x_f^{j,l}(t)$ variables are all 0 whenever t is not a feasible time for flight f to reach sector j; $x_f^{j,l}(t)$ variables are all 0 if t is the last feasible time for a flight to reach a sector j in its path

ll. 190-433 The problem constraints are defined, following the order of **Section 2.3**

ll. 437-446 The sense of the objective function, the objective function and the constraints are specified to the problem object defined at the beginning

l. 448 The model instance is written on a specific file format (`.lp`)

l. 450 A time limit for the solver is set

ll. 452-455 The solver is called on the problem object (the model) defined through the above passages

ll. 459-484 The solution is printed out on a `.txt` file. This file displays the total delay cost, the amount of ground holding delay and the amount of airborne delay. It also displays for each flights the actual departure and arrival times as well as the amount of ground-holding and airborne delay for that specif flight.

There are different ways in which a model can be created by CPLEX using Python. The one we choose works as follows. An ordered list containing all the problem variables *names* is defined (see line 150 in `ATFMP.py`). The objective function

is identified with a list containing all the coefficient of the linear objective function, where if a variable does not appear in the objective function its coefficient is 0 (see lines 152-164). Every problem constraint is identified by a string (e.g. `"cap_arr_airp_const(a,6)"`) and defined by a list, a number (a float) and a letter (a string). The list contains two different lists: the first contains the names of the variables involved in the constraint, the second contains the coefficients of the variables whose names are in the first list. The string (`"G"`, `"L"` or `"E"`) defines the sense of the constraint ($\geqslant, \leqslant$ or $=$). The float number indicates the right-hand side of the constraint. Before calling the solver on the problem object, we need to assign to the object the following attributes: the problem sense (minimize or maximize); the OF as defined above, the type of the variables (real, integers or binary) and the name of the variables; the constraint list (containing all the constraints lists described above), the constraints senses list (containing all the senses of the constraints), the rhs list (containing all the rhs of the constraints) and the list containing all the constraints names. Once that the CPLEX model object is defined through the above passages, the solver can be called on the object by the method `.solve()`

Chapter 7

Computational results

One key question that we want to address is: does our new model improve airspace utilization? In this chapter, we shall try to answer this question thanks to the computational results that we obtained from our own implementation of the model. For the implementation we have used IBM ILOG CPLEX Optimization Studio (version 12.6.0) using its Python (version 2.7) interface as the modelling language, on a PC with Intel® Core™ i5-2500 CPU 3.30GHz × 4 processor and Linux Ubuntu 16.04 LTS 64-bit OS. The python code that we implement is entirely reported in the **Appendix**, along with the description of the data file form. We solved each ATFM problem instance first using our model without separation constraints (7a), (7b) and (7c) and nominal sector capacities, then we solved them introducing to the model separation constraints with separation parameters t^O_{sep}=1, l^O_{sep}=1, t^V_{sep}=1, l^V_{sep}=1 and increasing nominal mean sector capacities by 30%, 45% and 60%. Furthermore, we imposed a running time limit of 1 800 seconds for small instances, while the limit was set to 3 600 seconds for bigger instances. According to the key question asked at the beginning of the present chapter, our purpose is to see whether or not some improvement is obtained introducing separation constraints and increasing sector capacities and to understand how the entity of this increase affects the solution. We have used 4 different instances. For the sake of simplicity, in these instances we assumed the

capacity parameters to be constant. In the following we show and analyze the results obtained for each instance.

7.1 Results on "super hexagon" instance

In this instances group we consider an hexagonal airspace composed by 13 sectors (a, b, c, d, e, f, g, h and i) and containing 6 airports (HAM, VCE, CDG, BER, FCO and LHR). This airspace is depicted in **Figure 7.1**. Each sector is vertically subdivided into 9 levels and we assumed that all levels are feasible for each flight. The maximum level variation between sectors is fixed for every flight to 3. The time periods are 44, representing 44 15-minutes time windows summing up to 11 hours. We tested the model over 5 different sets of flights of increasing size: 15, 20, 30, 60 and 75. For each instance discussed in this chapter we set for every flight the ground holding delay to 1 and the airborne delay to 3. We assumed for each instance an airports mean capacity of 5 aircrafts per time period, both for departure and arrival, while the sector basic mean capacity depends on the number of flights taken into account. As **Table 1** illustrates, we obtain a decisive improvement of the solution (except for the 30 flights case). The table rows correspond to the number of flights of the instance, while columns corresponds to the type of model used (without separation, with separation and capacities improved by 30%, 40%, 60%). For each instance are indicated

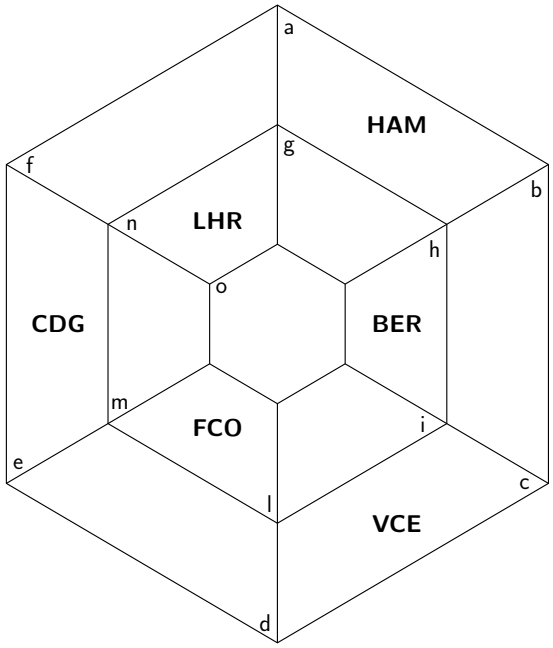


Figure 7.1: The super hexagon air space

the number of sectors and airports as well as the basic mean sector capacity and the airports departure and arrival mean capacity. For each numerical experiment is reported the total running time, the time to solve the root linear relaxation and the mean sector capacity of that instance. Finally, for each instance, the table shows the value of the objective function at the optimal minimum and the entity of the two types of delay: ground holding delay and airborne delay. Obviously this improvement increases as we increase the sector capacities. The entity of the solution improvement between the first and second column of the table is of:

- **15 flights:** +5.7%
- **20 flights:** +48.5%
- **30 flights:** $\begin{cases} -16\% \text{ lower bound} \\ -76\% \text{ upper bound} \end{cases}$
- **60 flights:** +42%
- **75 flights:** +54%

This results, with the exception of the 30 flights case, shows an improvement of the solution. This means that for these ATFM problem instances enhancing capacities by at least 30% allows a more flowing traffic, despite of the separation constraints.

super hexagon		DELAY TYPE	NO SEPARATION O.F.	NO SEPARATION GH	NO SEPARATION AB		SEPARATION, CAPACITY +30% O.F.	SEPARATION, CAPACITY +30% GH	SEPARATION, CAPACITY +30% AB		SEPARATION, CAPACITY +45% O.F.	SEPARATION, CAPACITY +45% GH	SEPARATION, CAPACITY +45% AB		SEPARATION, CAPACITY +60% O.F.	SEPARATION, CAPACITY +60% GH	SEPARATION, CAPACITY +60% AB	
						SOLVING TIME				SOLVING TIME				SOLVING TIME				SOLVING TIME
# SECTORS	13	**75 FLIGHTS**	76	76	0	294.07 sec.	35	35	0	31.21 sec.	24	24	0	14.60 sec.	**NO SOLUTION FOUND IN 3600.0 sec.**			
# AIRPORTS	6					ROOT 8.87 sec.				ROOT 5.02 sec.				ROOT 3.61 sec.				ROOT
BASIC MEAN CAPACITY	4					CAP. 4				CAP. 5.2				CAP. 5.8				CAP. 6.5
AIRPORTS MEAN CAPACITY (sep,arr)	5, 5																	
						SOLVING TIME				SOLVING TIME				SOLVING TIME				SOLVING TIME
# SECTORS	13	**60 FLIGHTS**	31	31	0	20.04 sec.	18	18	0	13.19 sec.	9	9	0	9.28 sec.	8	8	0	10.08 sec.
# AIRPORTS	6					ROOT 4.08 sec.				ROOT 2.21 sec.				ROOT 1.94 sec.				ROOT 1.87 sec.
BASIC MEAN CAPACITY	4.12					CAP. 4.2				CAP. 5,4				CAP. 6				CAP. 6.54
AIRPORTS MEAN CAPACITY (dep,arr)	5, 5																	
						SOLVING TIME				SOLVING TIME				SOLVING TIME				SOLVING TIME
# SECTORS	13	**30 FLIGHTS**	25	25	0	68.09 sec.	**UPPER BOUND:**		44	1800.11 sec.	**UPPER BOUND:**		42	1800.11 sec.	**UPPER BOUND:**		40	1800.10 sec.
# AIRPORTS	6					ROOT 6.04 sec.				ROOT 22.57 sec.				ROOT 23.17 sec.				ROOT 26.92 sec.
BASIC MEAN CAPACITY	3.1					CAP. 3.1	**LOWER BOUND:**		29	CAP. 4	**LOWER BOUND:**		27	CAP. 4.5	**LOWER BOUND:**		24	CAP. 5,1
AIRPORTS MEAN CAPACITY (dep,arr)	5, 5																	
						SOLVING TIME				SOLVING TIME				SOLVING TIME				SOLVING TIME
# SECTORS	13	**20 FLIGHTS**	33	33	0	1.47 sec.	17	17	0	1.19 sec.	14	14	0	1.18 sec.	14	14	0	1.14 sec.
# AIRPORTS	6					ROOT 0.29 sec.				ROOT 0.20 sec.				ROOT 0.22 sec.				ROOT 0.15 sec.
BASIC MEAN CAPACITY	2.6					CAP. 2.6				CAP. 3.4				CAP. **3.8**				CAP. 4,2
AIRPORTS MEAN CAPACITY (dep,arr)	5, 5																	
						SOLVING TIME				SOLVING TIME				SOLVING TIME				SOLVING TIME
# SECTORS	13	**15 FLIGHTS**	35	32	1	0.90 sec.	33	30	1	168.51 sec.	33	30	1	218.07 sec.	33	30	1	106.00 sec.
# AIRPORTS	6					ROOT 0.12 sec.				ROOT 2.69 sec.				ROOT 2.18 sec.				ROOT 1.26 sec.
BASIC MEAN CAPACITY	2.1					CAP. 2.1				CAP. 2.7				CAP. 3				CAP. 3.3
AIRPORTS MEAN CAPACITY (dep,arr)	5, 5																	

Table 1: Results on “super hexagon” instance

7.2 Results on "euro" instance

In this instances group we consider an europe-like airspace (imitiating the airspace comprising Italy, Croatia, Slovenia, Switzerland and Austria) composed by 7 sectors (a, b, c, d, e, f and g) each of which contains exactly one airports. The airports are: FCO, BRI, MXP, ZAG, LJU, BRN and VIE. As before, each sector is vertically subdivided into 9 levels and we assumed that all levels are feasible for each flight. The maximum level variation between sectors is fixed for every flight to 3. The time periods are 44. We tested the model over 4 different sets of flights of increasing size: 15, 30, 45 and 60. We assumed for each instance an airports mean capacity of 5.3 aircrafts per time period for departure and 3.9 aircrafts per time period for arrival, while the sector basic mean capacity is set to be 1.9 in every instance. As **Table 2** illustrates, in this case our model performs very poorly. The entity of the solution improvement between the first and second column of the table is:

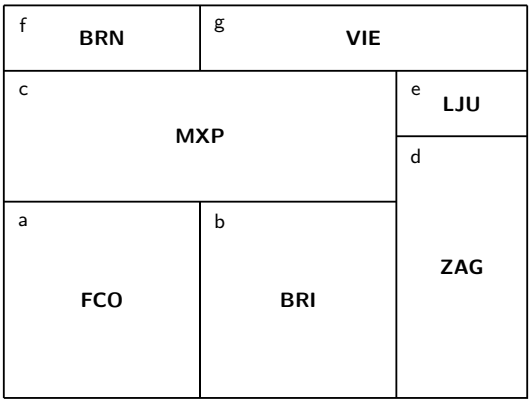


Figure 7.2: The euro air space

- **15 flights:** the solution changes from 0 to 6
- **30 flights:** -175%
- **45 flights:** $\begin{cases} -100\% \text{ lower bound} \\ -113\% \text{ upper bound} \end{cases}$
- **60 flights:** $\begin{cases} -32\% \text{ lower bound} \\ -144\% \text{ upper bound} \end{cases}$

These results look actually worse than in the previous instance, since the delay costs are more than double. One possible explanation is that in this instance a lot of flights cross in their path sector c and each flight which enters sector c prevents, because of opposing separation constraints, any other flights coming from an opposite edge to cross sector c, thus provoking a lot of congestion. Furthermore, the fact that the solution does not really change passing from column 2 to column 4 as the sector capacities increase, implies that the cause of delays in these columns does not come from sectors congestion but from separation constraints. In other words separation constraints are in this case stronger than the capacity constraints. Therefore, the trade off between increasing nominal sector capacities and introducing separation constraints is unfavorable, because we do not have any improvement by increasing capacities while separation constraints just provoke a solution worsening. Another important observation is that in these instances level distribution is not that efficient since most of the flight has a short path, during which they cannot reach an high level because of maximum level variation restrictions. For example, a flight from FCO to VIE with path (FCO, a, c, g, vie) cannot reach a level higher than 6, since its maximum level variation is set to 3.

euro			**NO SEPARATION**				**SEPARATION, CAPACITY +30%**				**SEPARATION, CAPACITY +45%**				**SEPARATION, CAPACITY +60%**			
		DELAY TYPE	F.O	GH	AB		F.O	GH	AB		F.O	GH	AB		F.O	GH	AB	
						SOLVING TIME				SOLVING TIME				SOLVING TIME				SOLVING TIME
# SECTORS	7	**60 FLIGHTS**	64	61	1	3.44 sec.	**UPPER BOUND:**		149	1800.05 sec.	**UPPER BOUND:**		126	1800.10 sec.	**UPPER BOUND:**		154	1800.04 sec.
# AIRPORTS	7					ROOT 0.36 sec.				ROOT 7.28 sec.				ROOT 7.55 sec.				ROOT 5.61 sec.
BASIC MEAN CAPACITY	1,9						**LOWER BOUND:**		81		**LOWER BOUND:**		80		**LOWER BOUND:**		77	
AIRPORTS MEAN CAPACITY (dep,arr)	5.3, 3.9					CAP. 1.9				CAP. 2.5				CAP. 2.8				CAP. 3.1
						SOLVING TIME				SOLVING TIME				SOLVING TIME				SOLVING TIME
# SECTORS	7	**45 FLIGHTS**	23	23	0	1.56 sec.	**UPPER BOUND:**		49	3606.4 sec.	49	49	0	5418.5 sec.	**UPPER BOUND:**		48	1824.37 sec.
# AIRPORTS	7					ROOT 0.18 sec.				ROOT 0.14 sec.				ROOT 1.58 sec.				ROOT 1.32 sec.
BASIC MEAN CAPACITY	1,9						**LOWER BOUND:**		46						**LOWER BOUND:**		42	
AIRPORTS MEAN CAPACITY (dep,arr)	5.3, 3.9					CAP. 1.9				CAP. 2.5				CAP. 2.8				CAP. 3.1
						SOLVING TIME				SOLVING TIME				SOLVING TIME				SOLVING TIME
# SECTORS	7	**30 FLIGHTS**	8	8	0	1.60 sec.	22	22	0	422.27 sec.	22	22	0	459.41 sec.	22	22	0	466.99 sec.
# AIRPORTS	7					ROOT 0.12 sec.				ROOT 0.43 sec.				ROOT 0.35 sec.				ROOT 0.39 sec.+
BASIC MEAN CAPACITY	1,9																	
AIRPORTS MEAN CAPACITY (dep,arr)	5.3, 3.9					CAP. 1.9				CAP. 2.5				CAP. 2.8				CAP. 3.1
						SOLVING TIME				SOLVING TIME				SOLVING TIME				SOLVING TIME
# SECTORS	7	**15 FLIGHTS**	0	0	0	0.79 sec.	6	6	0	4.45 sec.	6	6	0	5.08 sec.	6	6	0	5.78 sec.
# AIRPORTS	7					ROOT 0.02 sec.				ROOT 0.10 sec.				ROOT 0.08 sec.				ROOT 0.10 sec.
BASIC MEAN CAPACITY	1,9																	
AIRPORTS MEAN CAPACITY (dep,arr)	5.3, 3.9					CAP. 1.9				CAP. 2.5				CAP. 2.8				CAP. 3.1

Table 2: Results on “euro” instance

7.3 Results on "rectangle" instance

In this instances group we consider an airspace composed by 9 sectors (a, b, c, d, e, f, g, h and i) and 5 airports. The airports are: CDG, HAM, BER, FCO and VCE. Each sector is vertically subdivided into 9 levels and we assumed that all levels are feasible for each flight. The maximum level variation between sectors is fixed for every flight to 3. The time periods are 44. We tested the model over 3 different sets of flights of increasing size: 30, 45 and 50. We assumed for each instance an airports mean capacity of 9 aircrafts per time period for departure and 8.6 aircrafts per time period for arrival, while the sector basic mean capacity is set to be 5.3 in every instance. **Table 3** illustrates that also in this case our model shows really poor performances. The entity of the solution improvement between the first and second column of the table is of:

- **30 flights:** -3 800%
- **45 flights:** -745,5%
- **50 flights:** $\begin{cases} -616,7\% \text{ lower bound} \\ -1025\% \text{ upper bound} \end{cases}$

One can make the same arguments as above. The model probably performs poorly when flights path are short and therefore higher levels are not actually feasible.

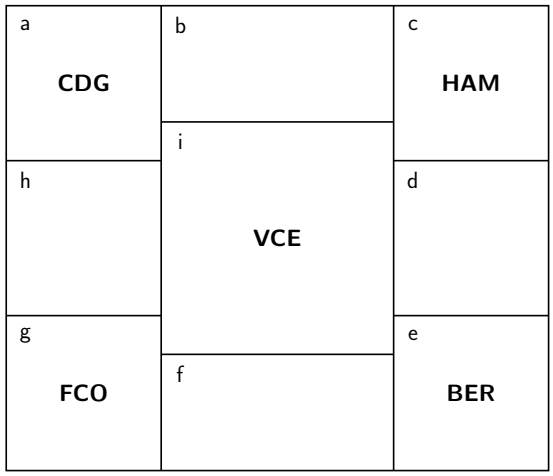


Figure 7.3: The rectangle air space

rectangle			NO SEPARATION				SEPARATION, CAPACITY +30%				SEPARATION, CAPACITY +45%				SEPARATION, CAPACITY +60%			
		DELAY TYPE	F.O	GH	AB		F.O	GH	AB		F.O	GH	AB		F.O	GH	AB	
						SOLVING TIME				SOLVING TIME				SOLVING TIME				SOLVING TIME
# SECTORS	9	**50 FLIGHTS**	12	12	0	4.05 sec.	**UPPER BOUND:**		135	3600.06 sec	**UPPER BOUND:**		124	3600.05 sec.	**UPPER BOUND:**		115	3600.04 sec.
# AIRPORTS	5					ROOT 0.33 sec.				ROOT 53.82 sec.				ROOT 52.67 sec.				ROOT 52.60 sec.
BASIC MEAN CAPACITY	5.3						**LOWER BOUND:**		86		**LOWER BOUND:**		86		**LOWER BOUND:**		86	
AIRPORTS MEAN CAPACITY (dep,arr)	9, 8.6					CAP. 5.3				CAP. 6.9				CAP. 7.8				CAP. 8.4
						SOLVING TIME				SOLVING TIME				SOLVING TIME				SOLVING TIME
# SECTORS	9	**45 FLIGHTS**	11	11	0	3.35 sec.	**UPPER BOUND:**		94	3600.07 sec.	93	93	0	2604.05 sec.	93	93	0	2115.35 sec.
# AIRPORTS	5					ROOT 0.28 sec.				ROOT 24.96 sec.				ROOT 27.00 sec.				ROOT 29.10 sec.
BASIC MEAN CAPACITY	5.3						**LOWER BOUND:**		93									
AIRPORTS MEAN CAPACITY (dep,arr)	9, 8.6					CAP. 5.3				CAP. 6.9				CAP. 7.8				CAP. 8.4
						SOLVING TIME				SOLVING TIME				SOLVING TIME				SOLVING TIME
# SECTORS	9	**30 FLIGHTS**	1	1	0	2.03 sec.	39	39	0	43.30 sec.	39	39	0	35.07 sec.	39	39	0	16.09 sec.
# AIRPORTS	5					ROOT 0.14 sec.				ROOT 3.28 sec.				ROOT 3.21 sec.				ROOT 4.87 sec.
BASIC MEAN CAPACITY	5.3																	
AIRPORTS MEAN CAPACITY (dep,arr)	9, 8.6					CAP. 5.3				CAP. 6.9				CAP. 7.8				CAP. 8.4

Table 3: Results on “rectangle” instance

7.4 Results on "donut" instance

In this last instances group we consider an airspace composed by 9 sectors (a, b, c, d, e, f, g, h and i) and 9 airports. The airports are: ATH, AMS, BER, CDG, CIA, DUB, EDI, GVA and IST. Each sector is vertically subdivided into 9 levels and we assumed that all levels are feasible for each flight. The maximum level variation between sectors is fixed for every flight to 3. The time periods are 44, representing 44 15-minutes time windows summing up to 11 hours. We tested the model over 2 different sets of flights of size 30 and 45. We assumed for each instance an airports mean capacity of 6.3 aircrafts per time period, both for departure and arrival, while the sector basic mean capacity is set to be 6 in one case and 3 in the other one. The results are illustrated by **Table 3**. The solution improvement between the first and second column of the table is in the two cases:

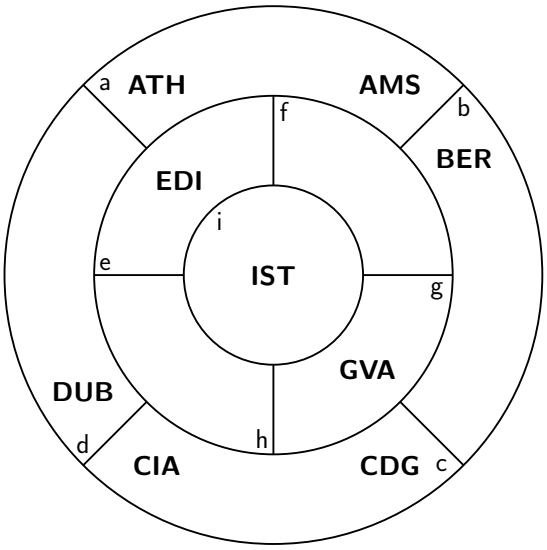


Figure 7.4: The donut air space

Basic mean capacity: 3

- **30 flights:** $\begin{cases} +17.5\% \text{ lower bound} \\ +1.6\% \text{ upper bound} \end{cases}$
- **45 flights:** $\begin{cases} +21,7\% \text{ lower bound} \\ -4.3\% \text{ upper bound} \end{cases}$

Basic mean capacity: 6

- **30 flights:** $\begin{cases} -716,6\% \text{ lower bound} \\ -783,3\% \text{ upper bound} \end{cases}$
- **45 flights:** $\begin{cases} -1\,400\% \text{ lower bound} \\ -2\,100\% \text{ upper bound} \end{cases}$

This example shows that the model can improve the solution when nominal capacities are low, while if capacities are high adding separation constraints is in comparison much more restrictive.

7.5 Discussion

Driven from the results discussed in the above sections, we conducted further experiments. These experiments showed the correctness of the hypothesis that short paths combined with low maximum level variations restricts flights possibilities to distribute. New experiments conducted on the two donut airspace cases without imposing any level maximum variation give the following results:

Basic mean capacity : 6

# flights	no separation	separation, cap +30%	separation, cap +45%	separation, cap +60%
30	1	4	4	4
45	6	10	10	10

Basic mean capacity : 3

# flights	no separation	separation, cap +30%	separation, cap +45%	separation, cap +60%
30	23	8	5	3
45	63	30-35	18-26	17-18

where two numbers separated by the hyphen (-) are solution's lower and upper bounds.
In this case the solution improvements between the first and the second column are in the two cases:

Basic mean capacity: 3

- **30 flights:** 65%
- **45 flights:** $\begin{cases} +52,4\% \text{ lower bound} \\ +44.4\% \text{ upper bound} \end{cases}$

Basic mean capacity: 6

- **30 flights:** -300%
- **45 flights:** -66.7%

donut			**NO SEPARATION**				**SEPARATION, CAPACITY +30%**				**SEPARATION, CAPACITY +45%**				**SEPARATION, CAPACITY +60%**			
		DELAY TYPE	F.O	GH	AB		F.O	GH	AB		F.O	GH	AB		F.O	GH	AB	
						SOLVING TIME				SOLVING TIME				SOLVING TIME				SOLVING TIME
# SECTORS	9	**45 FLIGHTS**	6	3	1	2.62 sec.	**UPPER BOUND:**		53	1800.04 sec.	**UPPER BOUND:**		61	1800.06 sec.	**UPPER BOUND:**		57	1800.11 sec.
# AIRPORTS	9					ROOT 0.25 sec.				ROOT 27.99 sec.				ROOT 28.69 sec.				ROOT 27.72 sec.
BASIC MEAN CAPACITY	6						**LOWER BOUND:**		49		**LOWER BOUND:**		49		**LOWER BOUND:**		49	
AIRPORTS MEAN CAPACITY (dep,arr)	6.3, 6.3					CAP. 6				CAP. 7.8				CAP. 8.6				CAP. 9.6
						SOLVING TIME				SOLVING TIME				SOLVING TIME				SOLVING TIME
# SECTORS	9	**30 FLIGHTS**	1	1	0	2.19 sec.	**UPPER BOUND:**		22	1800.09 sec.	**UPPER BOUND:**		22	1800.10 sec.	**UPPER BOUND:**		22	1800.05 sec.
# AIRPORTS	9					ROOT 0.17 sec.				ROOT 9.89 sec.				ROOT 9.59 sec.				ROOT 9.62 sec.
BASIC MEAN CAPACITY	6						**LOWER BOUND:**		15		**LOWER BOUND:**		15		**LOWER BOUND:**		15	
AIRPORTS MEAN CAPACITY (dep,arr)	6.3, 6.3					CAP. 6				CAP. 7.8				CAP. 8.6				CAP. 9.6
						SOLVING TIME				SOLVING TIME				SOLVING TIME				SOLVING TIME
# SECTORS	9	**45 FLIGHTS**	63	60	1	6.52 sec.	**UPPER BOUND:**		62	1800.06 sec.	**UPPER BOUND:**		57	1800.06 sec.	**UPPER BOUND:**		65	1800.04 sec.
# AIRPORTS	9					ROOT 0.59 sec.				ROOT 30.02 sec.				ROOT 30.28 sec.				ROOT 25.87 sec.
BASIC MEAN CAPACITY	3						**LOWER BOUND:**		52		**LOWER BOUND:**		48		**LOWER BOUND:**		49	
AIRPORTS MEAN CAPACITY (dep,arr)	6.3, 6.3					CAP. 3				CAP. 3.9				CAP. 4.3				CAP. 4,8
						SOLVING TIME				SOLVING TIME				SOLVING TIME				SOLVING TIME
# SECTORS	9	**30 FLIGHTS**	23	23	0	25.76 sec.	**UPPER BOUND:**		24	1800.04 sec.	**UPPER BOUND:**		22	1800.05 sec.	**UPPER BOUND:**		22	1800.08 sec.
# AIRPORTS	9					ROOT 5.40 sec.				ROOT 11.51 sec.				ROOT 14.10 sec.				ROOT 12.65 sec.
BASIC MEAN CAPACITY	3						**LOWER BOUND:**		18		**LOWER BOUND:**		16		**LOWER BOUND:**		15	
AIRPORTS MEAN CAPACITY (dep,arr)	6.3, 6.3					CAP. 3				CAP. 3.9				CAP. 4.3				CAP. 4,8

Table 4: Results on “donut” instance

Having relaxed level variation restriction has reduced the worsening of the solution for the case of 6 basic mean capacity and has increase the improvement of the solution for the case of 3 basic mean capacity. The deductions that we can make from these observation are quite natural and are the following: separation constraints effectiveness increases as the mean sector nominal capacity decrease. Separation effectiveness increases as the mean maximum level variation 1increases. Separation effectiveness increases as the mean length of flights' paths increases. One last and more relevant conclusion that we can draw is that potential opposing conflict separation constraints are maybe too restrictive and they hence need a revision.

In the next chapter final conclusions are drawn and possible improvements and future developments are discussed.

Chapter 8

Conclusions

In this thesis, we proposed an integer $0-1$ linear programming model to address the ATFM problem. The motivations to address such a problem, along with some general background on ATM, were first provided. We then gave some detail on integer linear programming theory and on modern available software. Next, we introduced the key ideas of our work and we described the specific characteristics of the ATFM problem that we wanted to include in our model as well as the strategies that we have adopted to address the problem. Afterwards, the mathematical details of the model were given. In particular, the main original contributions of our work have been accurately described: the formalization of the separation constraints, the introduction of simple trajectory constraints and the inclusion of two classes of valid inequalities. The validity of these inequalities has been then formally proven. Subsequently, we described the implementation of the model using the Python interface to the CPLEX solver. Finally, we reported the results of some computational tests on small instances, obtained using the implemented program.

The novel class of separation constraints that we introduced was proven to be actually effective. Conceptually, the main idea of our work was to control the air traffic flow in more detail then in the original model [5] from which our research began.

As a conclusion, our model allows to describe trajectory interactions more precisely, while it is still able to well capture macroscopic and general features of the problem. Hence, looking for a more accurate description of trajectories interactions, by means of decision variables similar to the one we used, can be a good and profitable direction to take for future research. This thesis has shown that such an approach is absolutely possible, since we manage to follow it successfully, without leaving the rich and powerful framework of integer linear programming. As a last consideration, we feel that some improvement of our model can be certainly achieved. We now suggest some possibilities that should be taken into account for future research.

After having conducted some experiments on small instances, one of the conclusion that we drew is that the class of constraints that we called opposing separation constraints is maybe too restrictive. In fact, the model performs well under some particular conditions (such as low nominal sector capacity and high maximum level variation). Otherwise the solution improvements can be negative. Our analysis suggests that the cause are opposing separation constraints, because among the separation constraints they are the most restrictive. These constraints then might need to be revised. One possible direction to future improvement of the model can be the relaxation of this separation requirements.

Future development of the model might include a reformulation of the objective function, e.g. including minimization of the number of flights exceeding a given time delay, as well as fairness consideration, like in previous works based on the same decision variables as the one we considered. It would be also easy to include speed control in the model, allowing more flexible solutions. Of particular interest would finally be the inclusion of rerouting in our model, hence describing complete 4D trajectories. In this case a possible objective function could also include penalization of alternative routes to the scheduled one.

As a conclusive suggestion, we would surely conduct more computational experiments in order to test the model behaviour on bigger instances. To this purpose, one should definitely take into consideration the inclusion in the model implementation of the valid inequalities exposed in **Section 5.9**. This could in fact reduce the running time of the implementation, allowing faster tests.

Appendix

In the following we include both an example of a data file and the entire code of the Python program that we wrote for the model implementation.

data_super_hexagon_15_NO_SEP.txt

```
## INSTANCE DESCRIPTION
# In this problem instance we test the model without separation constraints
# and with nominal capacities on the super_hexagon airspace:
#
#                              ----------------------
#                            /\a                    /\
#                           /f \         HAM       /b \
#                          /    \                  /    \
#                         /      \ ---------------/      \
#                        /       /\g             /\       \
#                       /       /n \           /h \       \
#                      /       /    \         /    \       \
#                     /       /      \-------/      \       \
#                    /       /  LHR  /o      \  BER  \       \
#                   /       /       /         \       \       \
#                  /-------/-------/           \-------\-------\
#                  \e      \m      \           /i      /c      /
#                   \       \       \         /       /       /
#                    \       \       \-------/       /       /
#                     \       \      /l      \      /       /
#                      \  CDG  \    /         \    /  VCE  /
#                       \       \  /    FCO    \  /       /
#                        \       \/-------------\/       /
#                         \      /d              \      /
#                          \    /                 \    /
#                           \  /                   \  /
#                            \/---------------------\/
#
## FLIGHTS
flights = ["f%d"%n for n in range(1,16)]

## AIRPORTS
airports = ["HAM","BER","VCE","FCO","CDG","LHR"]

## SECTORS
sectors = ["a","b","c","d","e","f","g","h","i","l","m","n","o"]

## LEVELS
levels = [0,1,2,3,4,5,6,7,8,9]

## TIME WINDOWS
times = [t for t in range(35)]
```

```
## PATHS
paths = {
"f1":("HAM","a","b","c","VCE"),
"f2":("HAM","a","g","h","i","c","VCE"),
"f3":("HAM","a","g","o","i","c","VCE"),
"f4":("HAM","a","f","e","CDG"),
"f5":("HAM","a","g","n","m","e","CDG"),
"f6":("HAM","a","g","o","m","e","CDG"),
"f7":("HAM","a","b","h","BER"),
"f8":("HAM","a","g","h","BER"),
"f9":("HAM","a","g","o","h","BER"),
"f10":("HAM","a","f","n","LHR"),
"f11":("HAM","a","g","n","LHR"),
"f12":("HAM","a","g","o","n","LHR"),
"f13":("HAM","a","g","o","l","FCO"),
"f14":("VCE","c","b","a","HAM"),
"f15":("VCE","c","i","h","g","a","HAM")}

## AIRPORTS CAPACITIES
airp_capacities = {
"FCO":(3.0,4.0),
"CDG":(5.0,4.0),
"VCE":(4.0,1.0),
"HAM":(5.0,6.0),
"LHR":(6.0,1.0),
"BER":(7.0,6.0)}

## SECTORS CAPACITIES
sect_capacities = {
"a":2.0,
"b":2.0,
"c":2.0,
"d":2.0,
"e":2.0,
"f":2.0,
"g":2.0,
"h":2.0,
"i":2.0,
"l":2.0,
"m":2.0,
"n":2.0,
"o":3.0}

## SECTORS CROSSING TIMES
crossing_times = {
"f1":(0,2,4,2,0),
"f2":(0,1,1,3,1,1,0),
"f3":(0,1,2,2,2,1,0),
"f4":(0,2,4,2,0),
"f5":(0,1,1,3,1,1,0),
"f6":(0,1,2,2,2,1,0),
"f7":(0,2,1,1,0),
"f8":(0,1,2,1,0),
"f9":(0,1,2,1,1,0),
"f10":(0,2,1,1,0),
"f11":(0,1,2,1,0),
"f12":(0,1,2,1,1,0),
"f13":(0,1,2,3,1,0),
"f14":(0,2,4,2,0),
"f15":(0,1,1,3,1,1,0)}
```

```
## (FIRST,LAST) FEASIBLE DEPARTURE TIMES
FL_dep_times = {
"f1":(1,1),
"f2":(2,2),
"f3":(2,20),
"f4":(2,20),
"f5":(3,20),
"f6":(4,20),
"f7":(3,20),
"f8":(3,20),
"f9":(1,20),
"f10":(1,20),
"f11":(3,3),
"f12":(2,20),
"f13":(2,20),
"f14":(3,20),
"f15":(2,20)}

## LAST FEASIBLE ARRIVAL TIMES
last_arr_times = {"f%d"%n:30 for n in range(1,16)}

## FEASIBLE LEVELS
feas_levels = {
"f1":{"HAM":[0],"a":[1,2,3,4,5,6,7,8,9],"b":[1,2,3,4,5,6,7,8,9],"c":[1,2,3,4,5,6,7,8,9],
    "VCE":[0]},
"f2":{"HAM":[0],"a":[1,2,3,4,5,6,7,8,9],"g":[1,2,3,4,5,6,7,8,9],"h":[1,2,3,4,5,6,7,8,9],
    "i":[1,2,3,4,5,6,7,8,9],"c":[1,2,3,4,5,6,7,8,9],"VCE":[0]},
"f3":{"HAM":[0],"a":[1,2,3,4,5,6,7,8,9],"g":[1,2,3,4,5,6,7,8,9],"o":[1,2,3,4,5,6,7,8,9],
    "i":[1,2,3,4,5,6,7,8,9],"c":[1,2,3,4,5,6,7,8,9],"VCE":[0]},
"f4":{"HAM":[0],"a":[1,2,3,4,5,6,7,8,9],"f":[1,2,3,4,5,6,7,8,9],"e":[1,2,3,4,5,6,7,8,9],
    "CDG":[0]},
"f5":{"HAM":[0],"a":[1,2,3,4,5,6,7,8,9],"g":[1,2,3,4,5,6,7,8,9],"n":[1,2,3,4,5,6,7,8,9],
    "m":[1,2,3,4,5,6,7,8,9],"e":[1,2,3,4,5,6,7,8,9],"CDG":[0]},
"f6":{"HAM":[0],"a":[1,2,3,4,5,6,7,8,9],"g":[1,2,3,4,5,6,7,8,9],"o":[1,2,3,4,5,6,7,8,9],
    "m":[1,2,3,4,5,6,7,8,9],"e":[1,2,3,4,5,6,7,8,9],"CDG":[0]},
"f7":{"HAM":[0],"a":[1,2,3,4,5,6,7,8,9],"b":[1,2,3,4,5,6,7,8,9],"h":[1,2,3,4,5,6,7,8,9],
    "BER":[0]},
"f8":{"HAM":[0],"a":[1,2,3,4,5,6,7,8,9],"g":[1,2,3,4,5,6,7,8,9],"h":[1,2,3,4,5,6,7,8,9],
    "BER":[0]},
"f9":{"HAM":[0],"a":[1,2,3,4,5,6,7,8,9],"g":[1,2,3,4,5,6,7,8,9],"o":[1,2,3,4,5,6,7,8,9],
    "h":[1,2,3,4,5,6,7,8,9],"BER":[0]},
"f10":{"HAM":[0],"a":[1,2,3,4,5,6,7,8,9],"f":[1,2,3,4,5,6,7,8,9],"n"
    :[1,2,3,4,5,6,7,8,9],"LHR":[0]},
"f11":{"HAM":[0],"a":[1,2,3,4,5,6,7,8,9],"g":[1,2,3,4,5,6,7,8,9],"n"
    :[1,2,3,4,5,6,7,8,9],"LHR":[0]},
"f12":{"HAM":[0],"a":[1,2,3,4,5,6,7,8,9],"g":[1,2,3,4,5,6,7,8,9],"o"
    :[1,2,3,4,5,6,7,8,9],"n":[1,2,3,4,5,6,7,8,9],"LHR":[0]},
"f13":{"HAM":[0],"a":[1,2,3,4,5,6,7,8,9],"g":[1,2,3,4,5,6,7,8,9],"o"
    :[1,2,3,4,5,6,7,8,9],"l":[1,2,3,4,5,6,7,8,9],"FCO":[0]},
"f14":{"VCE":[0],"c":[1,2,3,4,5,6,7,8,9],"b":[1,2,3,4,5,6,7,8,9],"a"
    :[1,2,3,4,5,6,7,8,9],"HAM":[0]},
"f15":{"VCE":[0],"c":[1,2,3,4,5,6,7,8,9],"i":[1,2,3,4,5,6,7,8,9],"h"
    :[1,2,3,4,5,6,7,8,9],"g":[1,2,3,4,5,6,7,8,9],"a":[1,2,3,4,5,6,7,8,9],"HAM":[0]}}

## MAXIMUM LEVEL VARIATIONS
max_level_var = {
"f1":{"HAM":3,"a":3,"b":3,"c":3},
"f2":{"HAM":3,"a":3,"g":3,"h":3,"i":3,"c":3},
"f3":{"HAM":3,"a":3,"g":3,"o":3,"i":3,"c":3},
"f4":{"HAM":3,"a":3,"f":3,"e":3},
"f5":{"HAM":3,"a":3,"g":3,"n":3,"m":3,"e":3},
"f6":{"HAM":3,"a":3,"g":3,"o":3,"m":3,"e":3},
"f7":{"HAM":3,"a":3,"b":3,"h":3},
"f8":{"HAM":3,"a":3,"g":3,"h":3},
"f9":{"HAM":3,"a":3,"g":3,"o":3,"h":3},
"f10":{"HAM":3,"a":3,"f":3,"n":3},
"f11":{"HAM":3,"a":3,"g":3,"n":3},
"f12":{"HAM":3,"a":3,"g":3,"o":3,"n":3},
"f13":{"HAM":3,"a":3,"g":3,"o":3,"l":3},
"f14":{"VCE":3,"c":3,"b":3,"a":3},
"f15":{"VCE":3,"c":3,"i":3,"h":3,"g":3,"a":3}}
```

```
## OPPOSING CONFLICT SETS
opposing_conflict = {
("a","b"):["c","h"],
("a","f"):["e","n"],
("a","g"):["n","h","o"],
("b","a"):["f","g"],
("b","c"):["d","i"],
("b","h"):["g","i","o"],
("c","b"):["a","h"],
("c","d"):["e","l"],
("c","i"):["h","l","o"],
("d","c"):["b","i"],
("d","e"):["f","m"],
("d","l"):["i","m","o"],
("e","d"):["c","l"],
("e","f"):["a","n"],
("e","m"):["l","n","o"],
("f","a"):["b","g"],
("f","e"):["d","m"],
("f","n"):["g","m","o"],
("g","a"):["b","f"],
("g","h"):["b","i"],
("g","n"):["f","m"],
("g","o"):["i","l","m"],
("h","g"):["a","n"],
("h","i"):["c","l"],
("h","o"):["l","m","n"],
("i","c"):["b","d"],
("i","h"):["b","g"],
("i","l"):["d","m"],
("i","o"):["g","m","n"],
("l","i"):["c","h"],
("l","m"):["e","n"],
("l","o"):["g","h","n"],
("m","e"):["d","f"],
("m","l"):["d","i"],
("m","n"):["f","g"],
("m","o"):["g","h","i"],
("n","g"):["a","h"],
("n","m"):["e","l"],
("n","o"):["h","i","l"],
("o","g"):["a"],
("o","h"):["b"],
("o","i"):["c"],
("o","l"):["d"],
("o","m"):["e"],
("o","n"):["f"]}

## CHASING CONFLICT SEPARATION
chase_time_separation = 0
chase_level_separation = 0

## OPPOSING CONFLICT SEPARATION
opp_time_separation = 0
opp_level_separation = 0

## GROUND HOLDING COSTS
grh_costs = {"f%d"%n:1.0 for n in range(1,16)}

## AIRBORNE DELAY COSTS
abd_costs = {"f%d"%n:3.0 for n in range(1,16)}
```

```
import cplex

problem = cplex.Cplex()

prob_name = input('Select problem data name: ')

f = open('data_%s.txt'%prob_name,'r')
exec(f.read())

## READING FUNCTIONS
def flight(var):
    if var[4] == ',': return var[2:4]
    elif var[5] == ',': return var[2:5]
    elif var[6] == ',': return var[2:6]
def sector(var):
    if var[4] == ',':
        if var[5] in sectors: return var[5]
        else: return var[5:8]
    elif var[5] == ',':
        if var[6] in sectors: return var[6]
        else: return var[6:9]
    elif var[6] == ',':
        if var[7] in sectors: return var[7]
        else: return var[7:10]
def level(var):
    if var[4] == ',':
        if var[6] == ',': return int(var[7])
        elif var[8] == ',': return int(var[9])
    elif var[5] == ',':
        if var[7] == ',': return int(var[8])
        elif var[9] == ',': return int(var[10])
    elif var[6] == ',':
        if var[8] == ',': return int(var[9])
        elif var[10] == ',': return int(var[11])
def time(var):
    if var[4] == ',':
        if var[6] == ',':
            if var[10] == ']': return int(var[9])
            elif var[11] == ']': return int(var[9:11])
            elif var[12] == ']': return int(var[9:12])
        elif var[8] == ',':
            if var[12] == ']': return int(var[11])
            elif var[13] == ']': return int(var[11:13])
            elif var[14] == ']': return int(var[11:14])
    if var[5] == ',':
        if var[7] == ',':
            if var[11] == ']': return int(var[10])
            elif var[12] == ']': return int(var[10:12])
            elif var[13] == ']': return int(var[10:13])
        elif var[9] == ',':
            if var[13] == ']': return int(var[12])
            elif var[14] == ']': return int(var[12:14])
            elif var[15] == ']': return int(var[12:15])
    if var[6] == ',':
        if var[8] == ',':
            if var[12] == ']': return int(var[11])
            elif var[13] == ']': return int(var[11:13])
            elif var[14] == ']': return int(var[11:14])
        elif var[10] == ',':
            if var[14] == ']': return int(var[13])
            elif var[15] == ']': return int(var[13:15])
            elif var[16] == ']': return int(var[13:16])
```

```
## UTILITY FUNCTIONS
def dep(var):
    return paths[flight(var)][0]
def arr(var):
    return paths[flight(var)][-1]
def len_flight(f):
    len_flight = 0
    for k in crossing_times[f]:
        len_flight += k
    return len_flight
def len_flight_var(var):
    return len_flight(flight(var))
def N_f(f):
    return len(paths[f])
def arr_times(f):
    arr_times=[]
    t = dep_times[f][0]+len_flight(f)
    while t in times:
        arr_times.append(t)
        t += 1
    return arr_times
def flight_time(f):
    return arr_times(f)[0]-dep_times[f][0]

## M CONSTANT DEFINITION
M = 100.0

## FEASIBLE DEPARTURE TIMES CONSTRUCTION
dep_times = {}
for f in flights:
    dep_times[f] = range(FL_dep_times[f][0], FL_dep_times[f][1]+1)

## FEASIBLE TIMES CONSTRUCTION
feas_times = {}
for f in flights:
    feas_times[f] = {}
    feas_times[f][paths[f][0]] = [int(dep_times[f][0]-1)] + [int(t) for t in dep_times[f
    ]]
    feas_times[f][paths[f][1]] = [int(dep_times[f][0]-1)] + [int(t) for t in dep_times[f
    ]]
    i = 1
    r = crossing_times[f][1]
    for j in paths[f][2:-1]:
        feas_times[f][j] = [t+r for t in feas_times[f][paths[f][1]]]
        i += 1
        r += crossing_times[f][i]
    feas_times[f][paths[f][-2]] += range(feas_times[f][paths[f][-2]][-1]+1,times[-1]+1)
    feas_times[f][paths[f][-1]] = feas_times[f][paths[f][-2]][crossing_times[f][-1]:]

## EDGES CONSTRUCTION
edges = []
for f in flights:
    for j in paths[f][1:-2]:
        i = paths[f].index(j)
        j_next = paths[f][i+1]
        v = (j,j_next)
        if v not in edges:
            edges.append(v)

## CHASING CONFLICT SETS CONSTRUCTION
chasing_conflict = {}
for e in edges:
    chasing_conflict[e] = []
    for f in edges:
        if f[1] == e[1] and f[0] not in opposing_conflict[e]:
            chasing_conflict[e].append(f[0])
```

```
## VARIABLES NAMES
x_variables = []
for f in flights:
    for j in paths[f]:
        for t in times:
            for l in feas_levels[f][j]:
                x_variables.append("x[%s,%s,%d,%d]" % (f, j, l, t))

y_variables = []
for f in flights:
    for j in paths[f][1:-1]:
        for l in feas_levels[f][j]:
            for t in times:
                    y_variables.append("y[%s,%s,%d,%d]" % (f, j, l, t))

u_variables = []
for e in edges:
    for l in levels:
        for t in times:
                u_variables.append("u[(%s,%s),%d,%d]" % (e[0], e[1], l, t))

variables = x_variables + y_variables + u_variables

## OBJECTIVE FUNCTION COEFFICIENTS
objective = [0.0 for v in variables]
for f in flights:
    for t in dep_times[f]:
        i = variables.index("x[%s,%s,0,%d]" % (f, paths[f][0], t))
        objective[i] += t*(grh_costs[f]-abd_costs[f])
        i_before = variables.index("x[%s,%s,0,%d]" % (f, paths[f][0], t-1))
        objective[i_before] -= t*(grh_costs[f]-abd_costs[f])
    for t in arr_times(f):
        k = variables.index("x[%s,%s,0,%d]" % (f, paths[f][-1], t))
        objective[k] += t*abd_costs[f]
        k_before = variables.index("x[%s,%s,0,%d]" % (f, paths[f][-1], t-1))
        objective[k_before] -= t*abd_costs[f]

## ADDITIVE CONSTANT IN OBJECTIVE FUNCTION
K_obj = 0
for f in flights:
    K_obj += (abd_costs[f]-grh_costs[f])*dep_times[f][0]-abd_costs[f]*arr_times(f)[0]

## AUXILIARY VARIABLES CONSTRAINT
aus_const = [[v for v in x_variables if time(v) <= feas_times[flight(v)][sector(v)][0]],
          [1.0 for v in x_variables if time(v) <= feas_times[flight(v)][sector(v)][0]]]
constraints = [aus_const]
rhs = [0.0]
constraint_names = ["aus_const"]
constraint_senses = ["E"]

## LAST FEASIBLE TIME CONSTRAINT
last_const = [[v for v in x_variables if time(v) == feas_times[flight(v)][sector(v)][-1]],
                [1.0 for v in x_variables if time(v) == feas_times[flight(v)][sector(v)][-1]]]
constraints.append(last_const)
k=0
for f in flights:
    k += float(len(paths[f]))
rhs.append(k)
constraint_names.append("last_const")
constraint_senses.append("E")
```

```
##[1] AIRPORTS DEPARTURE CAPACITY CONSTRAINTS
for a in airports:
    for t in times[1:]:
        air_const = [[],[]]
        for f in flights:
                if a == paths[f][0] and t in dep_times[f]:
                    air_const[0].append("x[%s,%s,0,%d]" % (f, a, t))
                    air_const[1].append(1.0)
                    air_const[0].append("x[%s,%s,0,%d]" % (f, a, t-1))
                    air_const[1].append(-1.0)
        if air_const != [[],[]]:
            constraints.append(air_const)
            rhs.append(airp_capacities[a][0])
            constraint_names.append("cap_dep_airp_const(%s,%d)"%(a,t))
            constraint_senses.append("L")

##[2] AIRPORTS ARRIVAL CAPACITY CONSTRAINTS
for a in airports:
    for t in times[1:]:
        air_const = [[],[]]
        for f in flights:
                if a == paths[f][-1] and t in arr_times(f):
                    air_const[0].append("x[%s,%s,0,%d]" % (f, a, t))
                    air_const[1].append(1.0)
                    air_const[0].append("x[%s,%s,0,%d]" % (f, a, t-1))
                    air_const[1].append(-1.0)
        if air_const != [[],[]]:
            constraints.append(air_const)
            rhs.append(airp_capacities[a][1])
            constraint_names.append("cap_arr_airp_const(%s,%d)"%(a,t))
            constraint_senses.append("L")

##[3] SECTORS CAPACITY CONSTRAINTS
for j in sectors:
    for t in times[1:]:
        cap_const = [[],[]]
        for f in flights:
            if j in paths[f]:
                j_next = paths[f][paths[f].index(j)+1]
                for l in feas_levels[f][j]:
                    cap_const[0].append("x[%s,%s,%d,%d]" %(f, j, l, t))
                    cap_const[1].append(1.0)
                for m in feas_levels[f][j_next]:
                    cap_const[0].append("x[%s,%s,%d,%d]" %(f, j_next, m, t))
                    cap_const[1].append(-1.0)
        if cap_const != [[],[]]:
            constraints.append(cap_const)
            rhs.append(sect_capacities[j])
            constraint_names.append("cap_sec_const(%s,%d)"%(j,t))
            constraint_senses.append("L")

##[4]  Y-VARIABLES CONSTRAINTS
for f in flights:
    for j in paths[f] [1:-1]:
        for l in feas_levels[f][j]:
            next_j = paths[f] [paths[f].index(j)+1]
            for t in times:
                y_const = [[],[]]
                y_const[0].append("y[%s,%s,%d,%d]" % (f, j, l, t))
                y_const[1].append(1.0)
                y_const[0].append("x[%s,%s,%d,%d]" % (f, j, l, t))
                y_const[1].append(-1.0)
                for m in feas_levels[f][next_j]:
                    y_const[0].append("x[%s,%s,%d,%d]" % (f, next_j, m, t))
                    y_const[1].append(1.0)
                constraints.append(y_const)
                rhs.append(0.0)
                constraint_names.append("y_const(%s,%s,%d,%d)"%(f, j, l, t))
                constraint_senses.append("G")
```

```
##[5] LEVEL UNIQUENESS CONSTRAINTS
for f in flights:
    for j in paths[f][1:-1]:
        for t in times :
            level_const = [[],[]]
            for l in feas_levels[f][j]:
                level_const[0].append("x[%s,%s,%d,%d]"%(f, j, l, t))
                level_const[1].append(1.0)
            constraints.append(level_const)
            constraint_names.append("level_const(%s,%s,%d)"%(f,j,t))
            constraint_senses.append("L")
            rhs.append(1.0)

##[6a] SECTORS STRONG CONNECTIVITY CONSTRAINTS
for f in flights:
    for j in paths[f][:N_f(f)-2]:
        i = paths[f].index(j)
        next_j = paths[f][i+1]
        for t in times:
            if t+crossing_times[f][i] in times:
                conn_const = [[],[]]
                for l in feas_levels[f][j]:
                    conn_const[0].append("x[%s,%s,%d,%d]"%(f, j, l, t))
                    conn_const[1].append(1.0)
                for m in feas_levels[f][next_j]:
                    conn_const[0].append("x[%s,%s,%d,%d]"%(f, next_j, m, t+
    crossing_times[f][i]))
                    conn_const[1].append(-1.0)
                constraints.append(conn_const)
                constraint_names.append("strong_conn_sec_const(%s,%s,%d)"%(f,next_j,t))
                constraint_senses.append("E")
                rhs.append(0.0)

##[6b] LEVELS  CONNECTIVITY CONSTRAINTS
for f in flights:
    for j in paths[f][:-2]:
        for t in feas_times[f][j][1:]:
            for l in feas_levels[f][j]:
                i = paths[f].index(j)
                next_j = paths[f][i+1]
                lev_conn_const = [[],[]]
                lev_conn_const[0].append("x[%s,%s,%d,%d]"%(f, j, l, t))
                lev_conn_const[1].append(1.0)
                for m in feas_levels[f][next_j]:
                    if m in range(l-max_level_var[f][j],l+max_level_var[f][j]+1):
                        lev_conn_const[0].append("x[%s,%s,%d,%d]"%(f, next_j, m, t+
    crossing_times[f][i]))
                        lev_conn_const[1].append(-1.0)
                constraints.append(lev_conn_const)
                constraint_names.append("lev_conn_const(%s,%s,%d,%d)"%(f,j,l,t))
                constraint_senses.append("L")
                rhs.append(0.0)

##[6c] SECTORS WEAK CONNECTIVITY CONSTRAINTS
for f in flights:
    j = paths[f][-2]
    for t in feas_times[f][j]:
        next_j = paths[f][-1]
        if t+crossing_times[f][-2] in feas_times[f][next_j]:
            conn_const = [[],[]]
            conn_const[0].append("x[%s,%s,%d,%d]"%(f, next_j, 0, t+crossing_times[f
    ][-2]))
            conn_const[1].append(1.0)
            for l in feas_levels[f][j]:
                if l in range(1,max_level_var[f][j]+1):
                    conn_const[0].append("x[%s,%s,%d,%d]"%(f, j, l, t))
                    conn_const[1].append(-1.0)
            constraints.append(conn_const)
            constraint_names.append("weak_conn_sec_const(%s,%s,%d)"%(f, next_j, t))
            constraint_senses.append("L")
            rhs.append(0.0)
```

```
##[7a]  CHASING SEPARATION CONSTRAINTS
for e in edges:
    j = e[1]
    for l in levels[1:]:
        for t in  times[1:]:
            chase_sep_const = [[],[]]
            for s in  range(t,t+chase_time_separation+1):
                for m in  range(l,l+chase_level_separation+1):
                    for k in chasing_conflict[e]:
                        for f in flights:
                            if k in paths[f]:
                                if j == paths[f][paths[f].index(k)+1]:
                                    if m in feas_levels[f][j] and s in feas_times[f][j
    ][1:]:
                                        chase_sep_const[0].append("x[%s,%s,%d,%d]"%(f, j
    , m, s))
                                        chase_sep_const[1].append(1.0)
                                        v = "x[%s,%s,%d,%d]"%(f, j, m, s-1)
                                        if v not in chase_sep_const[0]:
                                            chase_sep_const[0].append(v)
                                            chase_sep_const[1].append(-1.0)
                                        else:
                                            i = chase_sep_const[0].index(v)
                                            chase_sep_const[1][i]  -= 1.0
            if chase_sep_const != [[],[]]:
                constraints.append(chase_sep_const)
                constraint_names.append("chase_sep_const((%s,%s),%d,%d)"%(e[0],e[1], l,
    t))
                constraint_senses.append("L")
                rhs.append(1.0)

##[7c] OPPOSING SEPARATION CONSISTENCY CONSTRAINTS
for e in edges:
    j = e[0]
    j_next = e[1]
    for f in flights:
        if j in paths[f]:
            if j_next == paths[f][paths[f].index(j)+1]:
                for l in feas_levels[f][j_next]:
                    for t in times[1:-1]:
                        opp_sep_cons_const = [[],[]]
                        opp_sep_cons_const[0].append("u[(%s,%s),%d,%d]"%(e[0], e[1], l,
    t))
                        opp_sep_cons_const[1].append(-M)
                        for s in range(t,t+opp_time_separation+1):
                            for m in range(l-opp_level_separation,l+opp_level_separation
    +1):
                                for k in opposing_conflict[e]:
                                    for g in flights:
                                        if k in paths[g]:
                                            if j_next == paths[g][paths[g].index(k)+1]:
                                                if m in feas_levels[g][j_next] and s in
    feas_times[g][j_next][1:]:
                                                    opp_sep_cons_const[0].append("x[%s,%
    s,%d,%d]"%(g, j_next, m, s))
                                                    opp_sep_cons_const[1].append(1.0)
                                                    v = "x[%s,%s,%d,%d]"%(g, j_next, m,
    s-1)
                                                    if v not in opp_sep_cons_const[0]:
                                                        opp_sep_cons_const[0].append(v)
                                                        opp_sep_cons_const[1].append
    (-1.0)
                                                    else:
                                                        i = opp_sep_cons_const[0].index(
    v)
                                                        opp_sep_cons_const[1][i]  -= 1.0
                        if opp_sep_cons_const != [[],[]]:
                            constraints.append(opp_sep_cons_const)
                            constraint_names.append("opp_sep_cons_const((%s,%s),%s,%d,%d
    )"%(e[0],e[1], f, l, t))
                            constraint_senses.append("L")
                            rhs.append(0.0)
```

```
##[7b] OPPOSING SEPARATION CONSTRAINTS
for e in edges:
    j = e[0]
    j_next = e[1]
    for f in flights:
         if j in paths[f]:
             if j_next == paths[f][paths[f].index(j)+1]:
                for l in feas_levels[f][j_next]:
                    for t in times[1:-1]:
                        opp_sep_const=[[],[]]
                        opp_sep_const[0].append("y[%s,%s,%d,%d]"%(f, j_next, l, t))
                        opp_sep_const[1].append(1.0)
                        opp_sep_const[0].append("u[(%s,%s),%d,%d]"%(e[0], e[1], l, t))
                        opp_sep_const[1].append(1.0)
                        constraints.append(opp_sep_const)
                        constraint_names.append("opp_sep_const((%s,%s),%s,%d,%d)"%(e[0],
    e[1], f, l, t))
                        constraint_senses.append("L")
                        rhs.append(1.0)

##[8] ARRIVAL CONSTRAINTS
for f in flights:
    constraints.append([["x[%s,%s,%d,%d]"%(f, paths[f][-1], 0, last_arr_times[f])
    ],[1.0]])
    constraint_names.append("arr_const(%s)"%f)
    constraint_senses.append("E")
    rhs.append(1.0)

##[9] TIME CONNECTIVITY CONSTRAINTS
for f in flights:
    for j in paths[f]:
        for l in feas_levels[f][j]:
            for t in times[1:]:
                conn_temp = [[],[1.0, -1.0]]
                conn_temp[0].append("x[%s,%s,%d,%d]"%(f, j, l, t))
                conn_temp[0].append("x[%s,%s,%d,%d]"%(f, j, l, t-1))
                constraints.append(conn_temp)
                constraint_names.append("conn_temp_const(%s,%s,%d,%d)"%(f, j, l, t))
                constraint_senses.append("G")
                rhs.append(0.0)

##[10] BINARY TYPE VARIABLES CONSTRAINTS
my_ctype = ""
for v in variables:
    my_ctype += "B"

##########----CPLEX SOLVER----##########

problem.objective.set_sense(problem.objective.sense.minimize)

problem.variables.add( obj = objective,
                                    types = my_ctype,
            names = variables)

problem.linear_constraints.add( lin_expr = constraints,
                                                senses = constraint_senses,
                                                rhs = rhs,
                                                names = constraint_names)

problem.write("model_%s.lp"%prob_name)

problem.parameters.timelimit.set(7200.0)

try: problem.solve()

except CplexError:
    print('ERROR')
```

```
##########----PRINT SOLUTION TO FILE----##########

file = open('solution_%s.txt'%prob_name,'w')

file.write("SOLUTION STATUS: %s \n \n"%problem.solution.get_status_string())
if problem.solution.get_status_string() != "integer infeasible":
    file.write("SOLUTION:\n\n")
    file.write("Total delay cost: %f \n \n"%(K_obj+problem.solution.get_objective_value
    ()))
    actual_dep = {}
    tot_GH_del = 0
    tot_AB_del = 0
    for f in flights:
        for t in dep_times[f]:
            if problem.solution.get_values()[variables.index("x[%s,%s,0,%d]"%(f, paths[f
    ][0], t))]==1.0:
                file.write("Departure flight %s: %d    GH delay flight %s: %d \n"%(f,t,f
    ,t-dep_times[f][0]))
                actual_dep[f]=t
                tot_GH_del += t-dep_times[f][0]
                break
    file.write("\nTotal GH delay : %f \n \n"%tot_GH_del)
    for f in flights:
        for t in arr_times(f):
            if (problem.solution.get_values()[variables.index("x[%s,%s,0,%d]"%(f, paths[
    f][-1], t))]==1.0):
                file.write("Arrival flight %s: %d    AB delay flight %s: %d \n"%(f,t,f,t
    -actual_dep[f]-flight_time(f)))
                tot_AB_del += t-actual_dep[f]-flight_time(f)
                break
    file.write("\nTotal AB delay : %f \n \n"%tot_AB_del)

file.close()
```

References

[1] A. Agustı, A. Alonso-Ayuso, L. F. Escudero, C. Pizarro, et al. On air traffic flow management with rerouting. Part I: Deterministic case. *European Journal of Operational Research*, 219(1):156–166, 2012.

[2] A. Agustı, A. Alonso-Ayuso, L. F. Escudero, C. Pizarro, et al. On air traffic flow management with rerouting. Part II: Stochastic case. *European Journal of Operational Research*, 219(1):167–177, 2012.

[3] ATAG. `https://aviationbenefits.org/media/149673/abbb2016_eur.pdf`, 2016.

[4] D. Bertsimas, G. Lulli, and A. Odoni. An integer optimization approach to large-scale air traffic flow management. *Operations research*, 59(1):211–227, 2011.

[5] D. Bertsimas and S. Stock Patterson. The air traffic flow management problem with enroute capacities. *Operations research*, 46(3):406–422, 1998.

[6] D. Bertsimas and S. Stock Patterson. The traffic flow management rerouting problem in air traffic control: A dynamic network flow approach. *Transportation Science*, 34(3):239–255, 2000.

[7] M. Conforti, G. Cornuejols, and G. Zambelli. *Integer Programming*. Springer International Publishing, 2014.

[8] G. B. Dantzig and M. N. Thapa. *Linear programming 1: introduction*. Springer-Verlag, 1997.

[9] R. E. Bixby. A brief history of linear and mixed-integer programming computation. *Documenta Mathematica · Extra Volume ISMP*, pages 107–121, 2012.

[10] Eurostat. `https://ec.europa.eu/eurostat/web/transport/data/main-tables`. [Accessed on September 21, 2018].

[11] F. D. Fomeni, G. Lulli, and K. Zografos. An optimization model for assigning 4d-trajectories to flights under the tbo concept. In *Twelfth USA/Europe Air Traffic Management Research and Development Seminar (ATM2017). Seattle, Washington, USA June*, pages 26–30, 2017.

[12] F. Furini, C. A. Persiani, and P. Toth. The time dependent traveling salesman planning problem in controlled airspace. *Transportation Research Part B: Methodological*, 90:38–55, 2016.

[13] M. P. Helme. Reducing air traffic delay in a space-time network. In *Systems, Man and Cybernetics, 1992., IEEE International Conference on*, pages 236–242. IEEE, 1992.

[14] IATA. Tackling the european infrastructure crisis `https://airlines.iata.org/analysis/tackling-the-european-infrastructure-crisis`, 2016. [Accessed on September 21, 2018].

[15] ICAO. *Global Air Traffic Management Operational Concept*. Springer International Publishing, 2005.

[16] W. Kersten. *Global logistics management: sustainability, quality, risks*, volume 9. Erich Schmidt Verlag GmbH & Co KG, 2008.

[17] G. Lulli and A. Odoni. The european air traffic flow management problem. *Transportation science*, 41(4):431–443, 2007.

[18] A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[19] EUROCONTROL. Air Traffic Flow and Capacity Management (ATFCM) https://www.eurocontrol.int/articles/air-traffic-flow-and-capacity-management. [Accessed on September 21, 2018].

[20] EUROCONTROL. Pessimistic Sector Capacity Estimation https://www.eurocontrol.int/sites/default/files/library/026_Pessimistic_Sector_Capacity.pdf, 2003. page 3.

[21] EUROCONTROL. Long Term Forecast - Flight movements 2010 - 2030 https://www.eurocontrol.int/sites/default/files/publication/files/long-term-forecast-2010-2030.pdf, 2010.

[22] EUROCONTROL. Airspace Management Planning Chart ASM https://www.eurocontrol.int/sites/default/files/field_tabs/content/documents/nm/cartography/asm-desk-20oct2011.pdf, 2011.

[23] EUROCONTROL. Performance Review Report (PRR) 2011 https://www.eurocontrol.int/sites/default/files/publication/files/prr-2011.pdf, 2011.

[24] EUROCONTROL. EUROCONTROL ATM Lexicon https://ext.eurocontrol.int/lexicon/index.php/Air_Traffic_Flow_Management, 2015.

[25] EUROCONTROL. Demand Data Repository 2 (DDR2) https://www.eurocontrol.int/articles/ddr2-web-portal, 2018.

[26] FICO Xpress. https://www.fico.com/fico-xpress-optimization/docs. [Accessed on October 2, 2018].

[27] FICO Xpress Solver. https://www.fico.com/en/products/fico-xpress-solver. [Accessed on October 3, 2018].

[28] GUROBI. http://www.gurobi.com/products/features-benefits.
[Accessed on October 2, 2018].

[29] GUROBI. www.gurobi.com.
[Accessed on October 3, 2018].

[30] IBM. https://www.ibm.com/analytics/cplex-optimizer.
[Accessed on October 3, 2018].

[31] IBM. *CPLEX User's manual, version 12.1*, 2015.

[32] NATS. National Air Traffic Services
https://www.nats.aero/ae-home/introduction-to-airspace/, 2018.

[33] SKYbrary. https://www.skybrary.aero/index.php/Sectorisation, 2016.
[Accessed on September 23, 2018].

[34] SKYbrary. https://www.skybrary.aero/index.php/Separation_Standards, 2016.
[Accessed on September 23, 2018].

[35] SKYbrary. https://www.skybrary.aero/index.php/4D_Trajectory_Concept, 2017.
[Accessed on September 23, 2018].

[36] STATISTA. The Statistics Portal https://www.statista.com/statistics/193533/growth-of-global-air-traffic-passenger-demand/, 2018.
[Accessed on September 23, 2018].

[37] STATISTA. The Statistics Portal https://www.statista.com/statistics/564769/airline-industry-number-of-flights/, 2018.
[Accessed on September 23, 2018].

[38] STATISTA. The Statistics Portal https://www.statista.com/statistics/276222/number-of-airspace-sectors-in-europe-and-the-us/, 2018.
[Accessed on September 23, 2018].

[39] VATITA. Virtual Air Traffic ITA
http://www.vatita.net/drupal/airspace/fir-milano, 2010.

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
MATEMATICA