Università degli Studi di Trieste

DATA SCIENCE &
SCIENTIFIC COMPUTING

An integer programming model for a preference based inter airline slot trading

Author:
Andrea Gasparin

Supervisor:
Professor Lorenzo Castelli

ANNO ACCADEMICO 2019/2020

Contents

List of Figures

Abstract Italiano

Il progetto SESAR (Single European Sky ATM Research), promosso da Eurocontrol, da oltre dieci anni si occupa della ricerca di nuovi meccanismi e algoritmi per la riduzione dell'impatto economico causato dai ritardi dei velivoli. I ritardi sono generalmente causati da una Capacity Constraint Situations (CCSs): una temporanea riduzione di efficenza di una o più unità di controllo dello spazio aereo o degli areoporti. A tale scopo, SESAR ha promosso lo svuluppo del *Driven Prioritisation Process (UDPP)*, una procedura di *collaborative decision making* che ambisce ad aiutare le compagnie aeree nella gestione dei propri voli a cui è stato assegnato un ritardo. L'*UDPP* introduce svariati meccanismi per fornire una maggiore flessibilità alle compagnie per meglio identificare e definire le priorità dei propri voli. In questa tesi, abbiamo dimostrato che alcuni di questi meccanismi sono in realtà non necessari e che lo stesso risultato si può ottenere con una versione dell'algoritmo semplificata. Inoltre, abbiamo proposto un nuovo algoritmo che consente a ciascuna delle compagnie di ottimizzare i benefici economici apportati dall'*UDPP*. In aggiunta, abbiamo sviluppato un modello di programmazione intera che, in simbiosi con la filosofia propria dell'*UDPP*, estende ad esso la possibilità di realizzare scambi di slot fra diverse compagnie.

Abstract

In the context of SESAR (Single European Sky ATM Research), a huge effort has been made to find new mechanisms and algorithms in order to reduce flight delays and the consequent additional costs. Delay are generally caused by Capacity Constrained Situations (CCSs): an unexpected reduction of a resource ability to serve a certain number of flights per unit of time. When the number of flights exceeds this temporary lower capacity, some of them my be affected by a delay. Toward this aim, SESAR developed the User Driven Prioritisation Process (UDPP), a collaborative decision making procedure to support airlines in the management of their delayed flights, in order to reduce the delay cost impact. UDPP introduced several tools and features to help airlines in this prioritisation process. In this work, we first show that some of these features are actually redundant and that the same result could have been obtained, just applying some of them. We also develop an integer programming model to optimise the airlines usage of the UDPP features. In addition, in accordance with the UDPP philosophy, we develop another integer programming model that extends the possibility of inter-airline slot trading within the UDPP context.

Introduction

Motivations

In modern days the air traffic network became a consistent part of our lives. An exponentially increasing number of people use every day air transport for their business or holiday travels. As a side effect, this consistent growth of airspace demand, led unfortunately to a more difficult management of the network and consequently to a significant increase of delayed flights, which represents a major economical issue for both companies and passengers.
Since the early 2000s the American FAA (Federeal Aviation Administration) and EUROCONTROL, the European Organisation for the Safety of Air Navigation, have made a huge effort to find new mechanisms and algorithms in order to reduce delays and costs. This problem has been largely studied for decades also in academic literature and many solutions have been proposed, but a practical application of the existing results still struggles to be deployed. The main reason behind this fact lies in the intrinsic complexity of the task they are trying to address. In fact, any framework designed to tackle this problem should, on one hand consider the reduction of total network delay and, on the other, take into account the necessity to guarantee an equitable delay distribution among the different airlines and their need to be more involved in the delay allocation process. These requirements can sometimes be conflicting. Moreover, different flights have generally different economical values and this implies that the same delay might have a different cost impact. All the aforementioned concerns describe in very general terms what we will call the **delay impact problem**, which defines the subject of this work.

The context

When a flight is affected by a delay, the cause might be related to the flight itself or depending on a temporary resource deficiency, such as a sector (a delimited portion of the airspace) or an airport. The second scenario represents our case of interest, as the effects can potentially involve multiple **airspace users** (AUs) and its consequences might propagate to the entire network. Such deficiency is associated to the concept of *capacity*: the number of flights that a resource can

serve per unit of time. Unexpected events, such as bad weather, curfews, lack of visibility, closed runways, incidents, strikes..., might temporarily compromise the efficiency of a resource and, in order to keep guaranteeing safety, its capacity is reduced for a certain amount of time. This circumstance is called **constraint capacity situation** (CCS). When a CCS occurs, the resource might not be available for all the flights that originally requested it. If this happens, a flight has to be either rerouted or kept on the ground. In case of an airborne flight, it might be also applied an holding procedure, which consists in keeping an aircraft queuing on a circular route, located in certain specific airspace areas, until the resource is again available. For safety and economical reasons, this last procedure is generally tried to be avoided. How to handle the rescheduling or the rerouting of flights effected by a CCS, trying to limit the delay impact, is one of the hard tasks of the **Air Traffic Flow and Capacity Management** (ATFCM).

The delay impact problem in literature

One of the first mathematical formulation of this problem, has been made by Odoni [1987] who paved the way for all further model developments. Since then, the progress of the research in this field has been exponential, and many different approaches have been proposed, either stochastic, as for example the one of Terrab and Odoni [1993] or the one of Richetta and Odoni [1993], or deterministic. This work focus on this second approach and we will now illustrate a little bit more in detail some of the results which are particularly important to understand the specific issues that this thesis is trying to address. An interesting deterministic attempt to tackle this problem has been proposed by Bertsimas and Patterson [1998]. The linear programming model they developed, aimed to optimise the performance of the entire network considering either rescheduling and rerouting as available strategies to reduce the overall delay impact. This formulation requires a very detailed knowledge of the network configuration: in fact, for each flight, the model assumes to have at disposal an accurate description of the costs and the travel time of each flight sector. A further improvement toward the same aim has been presented by Bertsimas et al. [2011]. Despite the approach of these models ideally represents a very powerful and full comprehensive framework, it has a few drawbacks: the amount of variables needed to describe the entire system can make this formulation computationally intractable, and many of the parameters involved might be quite difficult to retrieve in a real scenario, especially at the level of accuracy required to obtain significant results. If on one hand, considering the entire network can be in certain circumstances very useful and mandatory in a future perspective, on the other, trying to analyse and improve the performance of a single resource is in many practical scenarios sufficient to guarantee a satisfactory improvement. In addition, at least in the European context, the rerouting procedure is less frequently deployed, due to the fact that the high level air traffic congestion is

concentrated in a quite limited airspace region. Therefore, many efforts have been made in order to reduce the delay impact, optimising the rescheduling of the flights involved in CCS, caused by a single resource. As long as in this context a schedule is nothing but an assignment of flights to some time slots, the delay impact problem can be seen as a problem of slot assignment. With this respect the OPTIFLOW model, proposed by Vossen and Ball [2006a], marked a significant step forward. In particular, Vossen and Ball introduced two very interesting features: the first is the interpretation of the slot assignment as an inter-airline slot swap mechanism, and the second is the possibility to identify, for each flight, a goal slot that the integer programming minimax algorithm they developed, will try to consider as the optimisation target of each flight. In our work, these concepts are of remarkable importance: the interpretation of the problem as an inter-airline slot swap dynamics allows to think at the slot assignment process as a sort of "slot trading" mechanism between several AUs. This perspective expands the horizon of the research and introduces the possibility of new strategies to tackle the delay impact problem. In addition, including in a mathematical model the idea of a goal slot signifies a direct involvement of the AUs in the slot assignment process. This aspect, in modern ATFCM, results crucial, toward the enhancement of the so called **collaborative decision making** (CDM). A further improvement of this model, has been proposed again by Vossen and Ball [2006b], with the formulation of the **at least-at most** (AMAL) model, which will be discussed in chapter 3. The *at least-at most* mechanism is a first attempt to formalise the inter-airlines trade dynamics, assuming that airlines might be interested in accepting a delay increase on some of their less important flights if they can obtain in return a reduction for their most strategical ones. A more refined version of the AMAL model has been proposed by Sherali et al. [2011], which exploits the graph representation of the slot assignment trading mechanism. The two models can achieve significant results, but they also show some practical limitations, that in chapter 3 will bee analysed in detail. As we mentioned at the beginning of this introduction, one of the obstacles characterising the delay impact problem is to handle the delay assignment without favouring or penalising any particular AU, and find a slot allocation process that distributes delays in an "equitable" manner. This concept is named **equity** or **fairness**, and it represents one of the main problem in the ATFCM context. A formulation of *equity* in mathematical terms has been attempted several times (we mention here Bertsimas and Gupta [2016]), but from the point of view of the airspace operators, the privileged strategy seems to be the development of new frameworks to enhance the CDM paradigm. Toward this aim, from 2004 in the context of SESAR programme (Single European Sky ATM Research), promoted by the European Uninon, the development of new mechanisms to favour the active role of the AUs in the slot allocation process under CCSs has been tried. In particular, it was introduced the possibility for the AUs to indicate their flight priorities within a framework called **User Driven Prioritisation Process** (UDPP). In addition, a consistent part of the effort has been made in order to optimise especially airport resources. One of the first important formulations of a UDPP algorithm, has been shown by

Pilon et al. [2016]. In the mechanism analysed, AUs priorities are used to determine a **flight delay apportionment** (FDA), a feature which allows an AU to distribute the amount of delay it must absorb among its flights. Moreover, a feature called **Selective flight protection** (SFP) has been developed. The SFP, allows an AU to gain a certain amount of credits, each time it decides to further increase the delay of one of its flights considered less important; these credits can be then spent, by the same AU, in order to reduce the delay of another of its more strategical flights. The UDPP framework and its features, have been periodically tested in several SESAR exercises, and further improvements have been developed. In its current version, that will be analysed in details in chapter 4, a new formulation of SFP has been developed, without credits, and the FDA has been replaced by the **Flight delay reordering** (FDR). This last feature is based on the concept of slot ownership, meaning that if a flight is assigned to a slot, that slot in certain sense belongs to the airline that owns the flight. The FDR allows each AU to reorder its flights within its own slots. In addition, one of the main characteristics of the UDPP, is that before determining a final slot assignment, it provides to the AUs a *what if* scenario, consisting in an approximate description of what they would achieve, applying the FDR and SFP features. This extra feature is a remarkable improvement in terms of flexibility, as it also allows airlines to adjust priorities on the spot, before the eventual commitment. UDPP achieved a wide consensus among the AUs and showed satisfactory experimental results. However, UDPP does not allow the possibility of inter-airline slot trading, and this signifies that there might still be room for improvements. Moreover, airlines with a limited number of flights involved in a CCS, the so called **Low Volume Operators in Constraint** (LVOC) might not be in the condition to fully exploit the features of UDPP. Toward this aim Ruiz et al. [2019] proposed a potential solution to extend to the LVOC the possibility to improve their situation, introducing a specific credit mechanism, named Flexible Credits for LVOCs (FCL). Despite some promising results, the main drawback of this solution is that it cannot guarantee some negative side effects, mainly an unwanted delay increas, on other AUs.

Currently, UDPP is implemented at Paris Charles de Gaulle and Frankfurt airports, and is planned for future implementations in Austria and Poland.

Structure of the document

This document is divided in five chapters:

- In **Chapter 1** we will set up the environment in which we are operating, providing a detailed description of the concept of CCS and its effects. We will also give an overview of the different methods used to estimate the costs caused by delays.
- In **Chapter 2** we will describe the First Planed First Served and the Compression algorithms, and we will use them as examples to introduce the delay impact problem.

- In **Chapter 3** we will analyse the AMAL model and the graph interpretation of the slot trading dynamics
- In **Chapter 4** the UDPP algorithm will be discussed in details. In particular, after the description of all the features of the UDPP framwork we will define the UDPPlocal and the UDPPmerge algorithm. In the second part of the chapter, we will show how UDPP might cause a positive impact even on AUs not participating to UDPP, but also how, in very exceptional situations, it might also create some negative side effects. In the last section, we will introduce an integer programming model, to find, for each airline, a priority assignment that optimises the UDPPlocal solution.
- In **Chapter 5** we will introduce an integer programming model, named **Inter-airlines slot offers provider** (ISTOP), that extends the possibility of inter-airline slot trading within the UDPP context. We will see how the formulation we propose aims to provide, as in the UDPP, a what if scenario consisting in a set of offers, that the AUs can decide whether to accept or refuse. We will also show how this model guarantees no negative side effects.
- In **Chapter 6** we will show some computational tests of the model presented in chapters 2, 3, 5 and we will discuss the results.
- In **Chapter 7** we will present the conclusions, the caveats and the possible future research developments.

Main contributions

With this work, we address three main tasks. We show that some of the additional features developed in the last UDPP version are actually redundant, and that the same result can be simply obtained with an appropriate usage of flight priority numbers, SFP and FDR. Then, we formulate an integer programming model to provide a priority assignment which optimise the UDPP local solution, assuming to have at disposal the function that accurately describes the delay impact of all flights of an AU. Finally, we develop a model that extends the possibility of inter-airline slot trading within the UDPP context, and in which the optimisation target are AUs preferences. In accordance with the CDM paradigm and the UDPP philosophy, the model, instead of a final schedule, aims to provide a what if scenario, consisting in a set of offers, that each AU can decide whether to accept or refuse. In addition, this particular framework provides a potential additional tool to reduce delay impact also for LVOCs. Also, due to a specific constraint included in the formulation, we guarantee that no negative side effect can be experienced by any AU.

Chapter 1

The ATM dinamycs

1.1 Increasing network congestion

As mentioned in the introduction, the air traffic volume is increasing. According to the data retrieved by Eurostat [2020] in the EU from 2013 to 2018 the number of passengers per year went from 838.920.866 up to 1.105.945.753 and the number of flights from 7.387.558 up to 8.323.657.

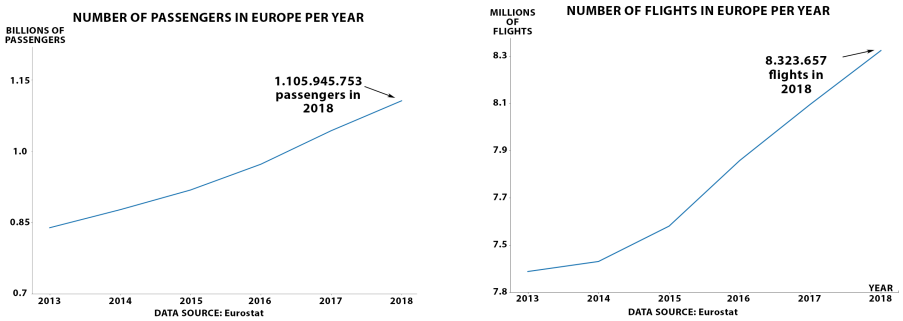


Figure 1.1: Air traffic volume over the years

Such growth, implies some potential issues for the **Air Traffic Management** (ATM), especially during periods of high congestion, which are typically between July and September: the main of these issues is represented by the flight delays. To further understand the reason of delay occurrences a basic introduction of the ATM mechanism is needed.

1.2 Airspace operations

The main actors in **Air Traffic Control** (ATC) are airports and sectors which are generally called **resources**. Airports manage all flight operations concerning

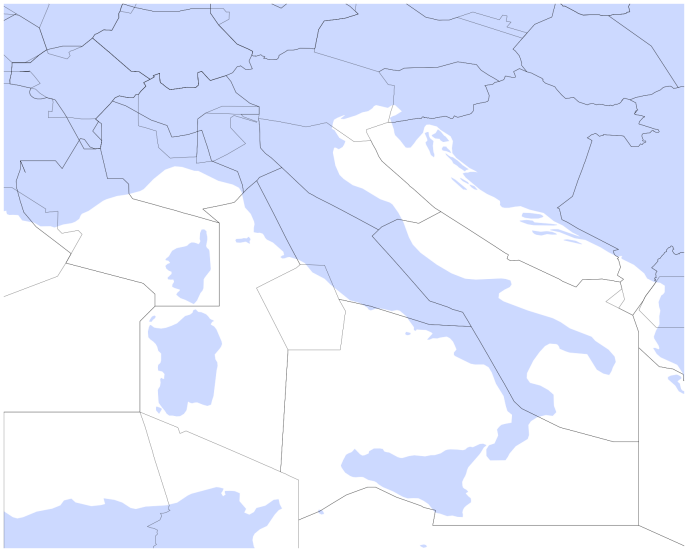


Figure 1.2: Sector configuration over Italy at 0.30 AM

aircraft on the ground, taking off and landing, for all the time they remain inside the surrounding control zone. Once a departing aircraft has left the airport control zone, it starts to follow its planned route until it enters into the destination airport control zone. To preserve safety during the en-route period of a flight, the entire airspace outside airport control zones, is partitioned into several sectors, each one managed by its own **Area Control Centre** (ACC). Their task is to keep constant radio contact with all the aircraft crossing their control area, in order to assist their navigation and guarantee that all safety regulations are respected, from the moment they enter into the flight sector until the moment they leave. Different sectors may also be located into the same geographical area, but referring to different altitude levels (flight levels). In order to maintain safety during congested periods, flight sectors' configuration may also variate over time. There are also many different possible configurations in a single day, depending on the predicted level of congestion.
Naturally, there is a limit to the number of flights that an airport or a flight sector can host. This limit depends on several factors, such as: equipment at disposal of the ATC and human resources deployed, number of runways in the case of airports and the size of the ACC in the case of sectors etc.... All these considerations lead to the following:

Definition 1 *The **capacity** is the maximum number of aircraft that can be accommodated in a given time period by the system or one of its components ATM Lexicon [2020].*

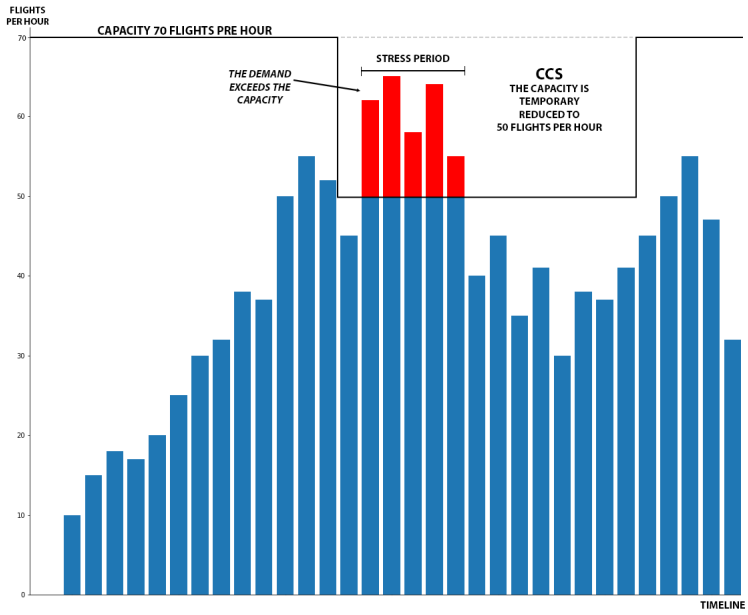


Figure 1.3: A CSS and its stress period

1.3 Delay occurrence

Fights schedules and routes are designed to never exceed any ATC capacity. They are generally defined far in advance but they can often be updated also a few weeks before the departure. Still, close to the operations day, many unexpected circumstances might occur: adverse weather conditions, visibility reduction, runway closures, instrumental issues, strikes, military operations and other unpredictable disservices. In order to guarantee safety, if any of these unexpected events affects a resource, the authorities might decide to decrease for a certain period of time the original number of flights that the resource was supposed to host.

Definition 2 *A **capacity constrained situation (CCS)** is a temporary capacity reduction of a resource.*

Especially during congested periods, when a CSS occurs, it might happen that the demand for a certain resource exceeds its reduced capacity. This scenario is the initial cause of what is called "hotspot", which we will now define in detail:

Definition 3 *In a CCS, the time window in which the demand for a certain resource exceeds its reduced capacity, is called **stress period**.*

Definition 4 *The time window between the end of a stress period and the time*

in which the exceeding demand is fully absorbed (all flights originally scheduled within the stress period are accommodated) is called ***recovery period****.*

Definition 5 *A* ***hotspot*** *is the entire time window that includes a stress period and the following recovery period [Pilon et al., 2016].*

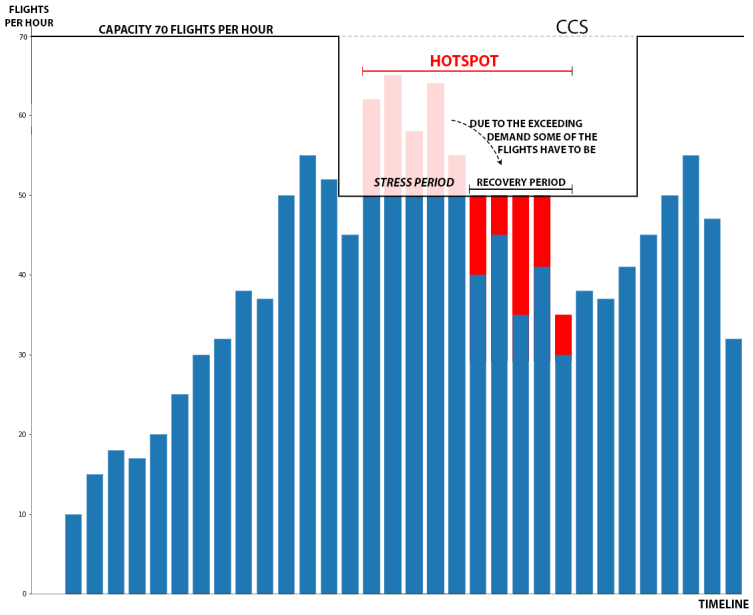


Figure 1.4: A graphical representation of an hotspot

If an hotspot occurs on a particular resource, some of the flights which were scheduled to be served within the stress period cannot be accommodated, as no resource can operate beyond its declared capacity. In addition, an aircraft is only allowed to take off if all the resources encountered on its route (airports and flight sectors) are available.
If the resource affected by a CCS is an airport, a possible solution is to apply a holding procedure to some of the flights, which consists in keeping an airborne aircraft in some dedicated airspace zones, generally located in the approaching area of the airport, until it receives the permission to land from the ACC. Despite this operation seems a natural solution, in practice it is performed just under certain occasional circumstances, and is tried to be avoided, as holding procedures cause additional fuel costs and in particular as it is considered unsafe to unnecessarily increase the number of airborne flights. If the exceeding capacity concerns a flight sector, another possibility is to reroute some of the flights scheduled to cross the sector during the critical period. However, especially when the entire air traffic network is congested, rerouting can sometimes

be an unfeasible solution, as a great number of sectors are working at near full capacity and there might be no other routes available. This last scenario is more likely to happen in Europe.
In practice, the most common procedure when a resource (either airport or flight sector) is affected by a CCS is to keep some of the involved flights on the ground. This, may lead to flight cancellations or to a postponed departure, until all necessary resources are again available. This latter case, is one of the main causes of flight delays. A hotspot over an airport is of particular interest as no rerouting strategies can be applied and because of the great number of actors potentially involved: departing flights, airport facilities and workers, auxiliary transports, ... This last scenario represents the environment in which this work is developed.

1.4 Delay allocation

There is also another way to describe the delay occurrence, and it requires the following:

Definition 6 *A **slot**, is an element of the partition of the time line. Given a certain resource, when a flight is assigned to a slot, that slot represents the only time window in which the flight is allowed to use the resource (to land in our case). Moreover, each slot is assumed to be available for only one flight.*

We will further assume that all slots are of the same duration, unless a CCS occurs. As long as each slot can host at most a single flight, the capacity can be alternatively interpreted as the number of slots per unit of time. This means that a capacity reduction implies an increase of the duration of the slots. If an airport can serve 60 flights per hour it means that the duration of the slots is 1 minute. If the capacity is reduced by 50%, the number of flights per hour drops down to 30, and the corresponding 30 slots will have a duration of 2 minutes. Ultimately, a flight f is affected by a delay, when the slot including its **expected arrival time** (ETA) is already occupied by another flight, and f has to be assigned to a later slot. In this case, we remark that the slot has to be a later one as a flight cannot land earlier than its ETA. Figure 1.5 shows a possible hotspot configuration from the slots perspective. However, this picture can also be misleading. In fact, all flights are originally scheduled for certain slots, and the hotspot implies that some of them have to be reassigned. The red arrows represent in a sense a possible reallocation, but, despite the fact that it seems the natural way to perform the reassignment, keeping the flights in the same order has been just an arbitrary choice. If one is interested for example in the **total delay**, which is the total sum of the delays of all flights, it might be that a different allocation would lead to a better result. In chapter 2 we will see that in this case things actually go in the way we expect, and that the assignment made in figure 1.5 minimises the total delay. The same reasoning holds, if one instead wants to minimise the total additional costs caused by the delays, which definitely represents a more challenging task. How to handle the

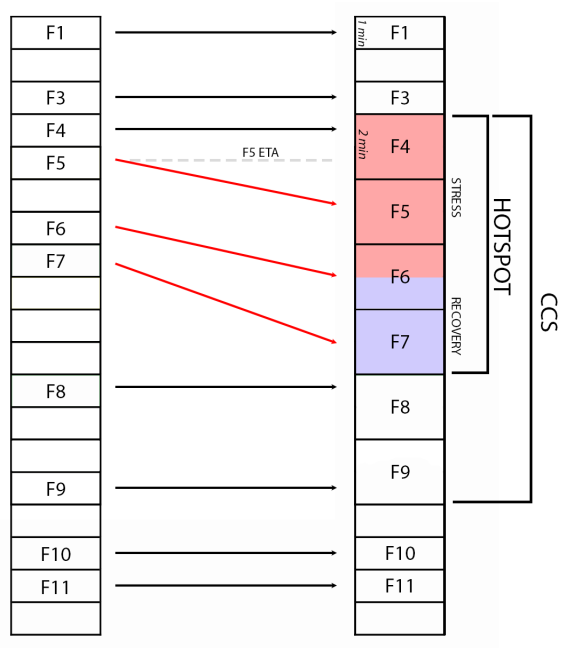


Figure 1.5: A graphical representation of an hotspot caused by a 50% capacity reduction, from the slots point of view

slot allocation process in a CCS will be the scope of this work, and we will refer to it as the **delay impact problem**. Before starting to analyse the possible solutions, we first need to understand how additional costs caused by delays are estimated.

1.5 Costs estimation

To increase the costs of a flight when a delay occurs, many aspects can contribute: fuel, maintenance, crew, passengers, reputation, etc.. An accurate estimation for a particular flight in a specific situation represents an hard task, sometimes even for the AUs themselves. However, a statistical description of the average behaviour of these costs has been widely studied and a very detailed research has been promoted by EUROCONTROL and realised by Cook and Tanner [2011] and Cook and Tanner [2015]. In their reports, many costs types and causes are analysed and differentiated by aircraft type.
For the scope of this work such level of accuracy is not necessary, but an important remark is that in general terms the costs can be seen as a non linear function of the delay. In particular, to approximate the delay impact for a specific flight f in we will generally use the formula:

$$\mathcal{C}(c_f, d_f) = \frac{c_f \cdot d_f^2}{2} \tag{1.1}$$

where c_f represents the cost per minute of delay of flight f and d_f its minutes of delay. This quadratic model, which has been already used in literature (*see* Ruiz et al. [2019]), is certainly a strong simplification, but as long as this work aims to provide just a first exploration of potential new models, for our purposes it is a sufficient approximation.

Chapter 2

FPFS

2.1 First planned first served

The procedure currently used by ATM to menage CCS, consists, first, in an initial application of the **First Planed First Served (FPFS)**. FPFS is a simple algorithm that reallocates all flights under CCS into the new time table dictated by the CCS, preserving the initial order:

Algorithm 1: FPFS

F = flights under CCS
sort(F) by increasing scheduled time of arrival
for *f in F* **do**
  assign flight f to the first unassigned slot
end

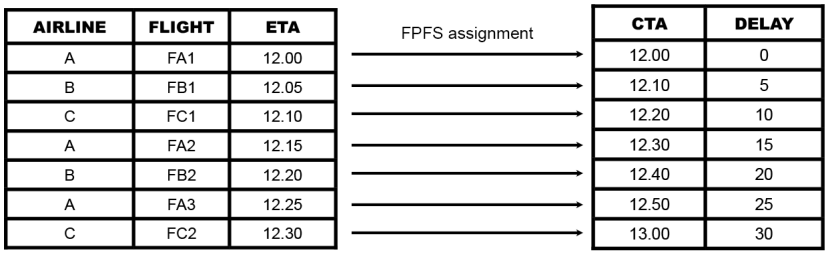


Figure 2.1: FPFS assignment

Even though the solution might look naive, FPFS ensures significant delay reduction. In fact, the following theorem, the proof of which can be found in Castelli et al. [2011] holds:

Theorem 1 *Over all possible slot allocations, FPFS guarantees optimality in terms of total delay reduction.*

FPFS remains also important from another perspective. In facts, it is generally considered a $fair$ delay assignment procedure by AUs, as preserving the initial flights order, it guarantees a natural equitable distribution of delays among the airlines with no particular selective advantage or disadvantage to any one. On the other hand, the delay constitutes just a partial metrics for AUs: the amount of passengers, the logistics, the possibility of providing connections with other flights, constitute just a few factors which show that different flights might have different economical value. Therefore if the same delay is assigned to two different flights, it might have a different economical impact. Moreover, as we earlier discussed in section 1.5, the cost increase can be approximately represented as a function of the delay.
Despite FPFS provides an optimal solution toward delay reduction, it does not guarantee similar results from the costs point of view. This aspect can be easily shown with an example:

Example 1 *Let us now assume to know the economical value of the flights depicted in figure 2.2, and that we are interested in analysing the delay impact, under the assumption that costs are given by the equation 1.1 defined in section 1.5:*

$$quadratic = flight\ value \cdot (delay)^2$$

After FPFS assignment, we would have the following situation:

AIRLINE	FLIGHT	COST PER MINUTE	ETA
A	FA1	6	12.00
B	FB1	3	12.05
C	FC1	1	12.10
A	FA2	2	12.15
B	FB2	3	12.20
A	FA3	10	12.25
C	FC2	8	12.30

FPFS assignment

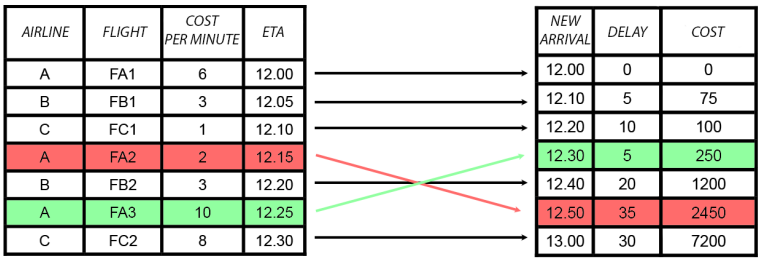
NEW ARRIVAL	DELAY	COST
12.00	0	0
12.10	5	75
12.20	10	100
12.30	15	450
12.40	20	1200
12.50	25	6250
13.00	30	7200

Figure 2.2: FPFS assignment

If we focus on airline A, we notice that the total delay impact amounts to:

$$total\ costs\ of\ airline\ A = \sum_{flight \in A} flight\ value \cdot (delay)^2 = 6700,$$

but if we now suppose to swap flights $FA2$ and $FA3$ we would then obtain:



total costs of airline A = 2700,

Figure 2.3: Swap of flights $FA2$ and $FA3$

which provides a significant cost reduction.
As mentioned before, the true dependency between costs increase and delay, might be much more complex than what we are considering in this example, but the same reasoning can be applied to any case.

In the last example we have shown how an AU, airline A in this case, might find convenient to change the order of its scheduled flights during a CCS after FPFS has been applied. It is important also to notice that the schedule changes operated by airline A do not effect any other AU, as all the swaps occur just between flights and slots owned by A.

Definition 7 *Any change of flight order within the same AU defines an **intra-airline operation**.*

Definition 8 *If an intra-airline operation does not effect any other AUs, it will be called an operation of **type** $\mathcal{I}$.*

2.2 Compression

Let us now analyse a frequent scenario which might take place during a CSS.

Example 2 *Referring to Example 1, suppose that the delay imposed to flight FB1, makes no longer worth for airline B to execute it. This might be caused by the fact that flight FB1 was supposed to serve a connection with another flight and the delay does not guarantee the passengers to be on time for it. So airline B finds now convenient to cancel FB1. If we look at the schedule, we can notice that the cancellation of flight FB1 allows all consequent flights to be shifted one position up and therefore gain a little delay reduction.*

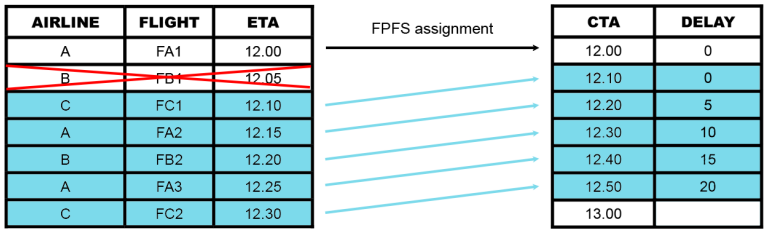


Figure 2.4: Potential scenario after $FB1$ cancellation

Despite not being particularly significant for each single airline, in terms of global delay amount this shift would provide a consistent improvement. In facts all flights consecutive to $FB1$ gain a delay reduction of just 5 minutes, where the global delay (excluding flight $FB1$), from 105 minutes, drops down to 50 minutes.

However ATM authorities noticed that airlines were generally hesitant in reporting cancellations. In order to encourage airlines to timely and accurately report any delay or cancellation, ATM developed the **Compression** algorithm, which referring to Example 2 can be explained in the following way:

- airline B notifies flight $FB1$ cancellation, so $12:10$ slot is now free
- in return, the following consequent flight owned by airline B within the regulation, flight $FB2$, it is tried to be assigned to the freed slot, in this case the $12:10$ one. If this assignment was possible, the algorithm would have finished.
- slot $12:10$ is earlier than flight $FB2$ ETA, so the assignment cannot take place. Then, all flights, up to $FB2$ earliest feasible slot $(12:20)$ are pushed up of one position, in this case just $FC1$.
- $12:20$ slot is now free and $FB2$ can be assigned to it
- $FB2$ initial slot is now free, so all later flights can be pushed one position up, in this case $FA3$ and $FC2$

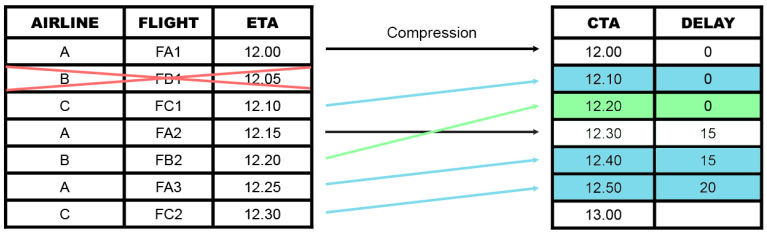


Figure 2.5: Compression algorithm applied to Example 1

After this introduction of the compression mechanism we can give a detailed description of the algorithm:

Algorithm 2: Compression

```
C = sets of free slots;
F = flights under regulation;
sort(F) according to the schedule produced by FPFS ;
for c in C do
    f* = cancelled flight, currently assigned to c;
    F.remove(f*);
    a = airline that owns slot c;
    endOfSchedule = false;
    while endOfSchedule == false do
        f = first consequent flight of f* owned by a;
        if f! = null and is assignable to c then
            assign f to c;
        else
            f = first consequent flight of f* owned by any other airline;
            if f! = null and f is assignable to c then
                assign f to c;
                f* = f;
                c = slot previously owned by f;
            else
                endOfSchedule = true;
            end
        end
    end
end
```

An important **remark**. This mechanism can be seen as an intra-airline operation: in example 2 for instance, airline B is cancelling a flight and thanks to *compression* algorithm, it receives an earlier slot for one of its other flights in return. The difference from example 1 is that in this case the operation has a positive effect also on other AUs.

Definition 9 *If an intra-airline operation has a positive effect on any other AUs, it will be called an operation of* ***type*** $\mathcal{C}$.

2.3 Trade opportunities

Now, let's look again at Example 1. One can notice that A is the only airline that can improve its situation applying an operation of type $\mathcal{I}$. In fact, airline C might also have interest in reducing flight $FC2$ delay, maybe accepting a later position in the schedule for flight $FC1$, but $ETA(FC2) = 12:30$, which makes impossible to swap its position with flight $FC1$. But, if we now allow

airline C to barter its slots with other airlines, as shown in example 3, we might obtain a further improvement of the overall situation compared to the one seen in Example 1.

Example 3 *In example 1 the cost reduction obtained by airline A swapping flights $FA2$ and $FA3$, amounts to:*

$$\textit{costs reduction of airline } A = -4000.$$

But, if we look at the situation after the schedule reshuffle depicted in figure 2.6, we obtain:

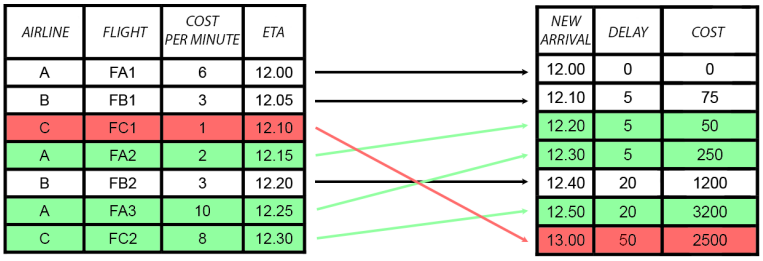


Figure 2.6: Slot bartering example

airline A:

$$\textit{costs reduction of airline } A = -6400,$$

airline C:

$$\textit{costs reduction of airline } C = -1600,$$

which represents a further improvement compared to the one provided by Example 1

In Example 3 the schedule reshuffle can be seen as a slot trading between different airlines, and this leads to the:

Definition 10 *a slot exchange involving multiple airline defines an **inter-airline operation** and we will refer to it as operation of **type** $\mathcal{T}$.*

Chapter 3

AMAL

3.1 A naive approach

A first attempt to tackle the delay impact problem, can be done simply trying to directly minimise the overall costs. To do this, the only requirement is the following:

Assumption 1 *The true cost function $\mathcal{C}$ exists and is the same for all airlines.*

The statement represents a quite unrealistic assumption, but allows us to point out some typical issues that one might encounter when dealing with the delay impact problem. To develop our first simple model we need to introduce:

- N, the number of slots within the hotspot
- $\mathcal{A}$, the set of AUs involved
- F, the set of flights involved
- S the set of all slots, $S := \{1, ..., N\}$
- $ETA(i)$, the function returning the slot corresponding to the ETA of fight i
- d_{ij} the delay of flight i if assigned to slot j
- $\mathcal{C}(d_{ij})$, cost function, with $\mathcal{C} : \mathbb{R}^+ \to \mathbb{R}^+$

and define the decision variables:

- $x_{ij} \in \{0, 1\}$, flight originally assigned to i is assigned to j

Constraint 1 *All flights have to be assigned:*

$$\sum_{j \in S} x_{ij} = 1 \quad \forall i \in F$$

Constraint 2 *A slot can host at most one flight:*

$$\sum_{i \in F} x_{ij} <= 1 \quad \forall j \in S$$

Definition 11 *The target is to minimise the overall costs:*

$$OBJ := \min \sum_{i \in F, j \in S} x_{ij} \cdot \mathcal{C}(d_{ij})$$

This first formulation, represents a very naive approach as, despite it guarantees optimality in terms of cost reduction, it does not consider how this reduction is distributed among the airlines. This issue, introduces a very important and difficult aspect in ATM: **equity** or **fairness**. Lets see an example, shown in figure 3.1, of a solution provided by the previous model:

FPFS

FLIGHT	COST PER MINUTE
A0	8
C1	1
B2	6
A3	5
B4	10
A5	7
B6	4
C7	2
B8	10
C9	3
A10	21
A11	9
B12	11
C13	2
A14	15

NAIVE

FLIGHT	COST PER MINUTE
A0	8
B2	6
B4	10
A3	5
A5	7
B8	10
A10	21
A11	9
B12	11
A14	15
B6	4
C9	3
C7	2
C13	2
C1	1

AIRLINE	INITIAL TOTAL COST	FINAL TOTAL COST
A	6349	513
B	2552	1000
C	680	2152
TOTAL	9581	3665

Figure 3.1: Solution provided by the naive model

Despite the very significant total cost reduction, the benefits are shared between airlines A and B. On the other hand, airline C is effected by a substantial negative impact and we can easily imagine that it would not consider this solution as *fair*. In our context, the main difficulty in dealing with *Equity* is the fact that it is hard to accurately determine what is its actual meaning. However, for our analysis a kind of initial reference point for this concept is needed, and in very general terms, we can state the following:

Definition 12 *A mechanism to reduce delay impact is considered **fair** when it is not biased toward privileging or penalising any particular airline.*

This definition is certainly loose, and allows many interpretations, but up to a certain extent it grasps the fact that, ultimately, all airlines might intend *fairness* in a slightly different way, and this makes impossible to provide a unique and rigorous mathematical description. Still, some statements and considerations can be made and we will see which strategies can be adopted to work around this issue.
Going back to the example of figure 3.1, we can observe that the poor result in terms of equity, is not actually surprising: the model is asked to minimise the total cost, therefore it will be biased to privilege flights with high costs. As long as flights of airline C have all lower costs, compared to the others, the model ultimately assigned them to the latest positions in the schedule.
A first step to improve the model, can be made, if we add the:

Assumption 2 *No airline would accept to be penalised by a change of schedule w.r.t the FPFS one.*

To include this principle in the model, we can just add the following:

Constraint 3 *For any airline, the resulting total delay cost, caused by the new slot allocation, has to be lower or equal to the one produced by $FPFS$:*

$$\sum_{i \in F_a,\, j \in S} x_{ij} \cdot \mathcal{C}(d_{ij}) \leq \sum_{i \in F_a} \mathcal{C}(d_{ii}) \quad \forall a \in A \tag{3.1}$$

The solution obtained with the introduction of the new constraint, is shown in figure 3.2 .

FPFS

FLIGHT	COST PER MINUTE
A0	8
C1	1
B2	6
A3	5
B4	10
A5	7
B6	4
C7	2
B8	10
C9	3
A10	21
A11	9
B12	11
C13	2
A14	15

MAX REDUCTION

FLIGHT	COST PER MINUTE
A0	8
B2	6
B4	10
C1	1
A5	7
B8	10
A10	21
C9	3
B12	11
C7	2
A14	15
A11	9
A3	5
C13	2
B6	4

AIRLINE	INITIAL COST	FINAL COST
A	6349	3981
B	2552	2152
C	680	680
TOTAL	9581	6813

Figure 3.2: Solution obtained by the *max reduction*

The total cost reduction is not as consistent as in the case without the additional constraint (figure 3.1), but, as expected, this time no negative impact is experienced by airline C. On the other hand, still no benefit is provided to C either. As before, the reason is that, even with the introduction of the additional constraint, the model is still biased to privilege high cost flights. However, despite this limitation, the model will turn out quite useful to compare the results provided by other models. In fact, if we look at the assumption 2 as a basic and mandatory constraint, the solution of this model can be considered as an upper bound for delay impact reduction. From now on, we will refer to this model as **max reduction**. In other words, if a model provides a solution that is considered *fair* under certain criteria, but its total impact reduction is far from the *max reduction* one, it might signify that there is still room for improvement. On the other hand, if the model performance is close to the *max reduction*, it can be interpreted as a satisfactory result.

We will give now an high level overview of two of the most important integer programming formulations of the delay impact problem with particular focus on their underlying trade interpretation, introducing some of the ideas that we will develop in the model presented in chapter 5.

3.2 Vossen Ball model

We have seen how *equity* represents a crucial aspect in this context, but its blurry nature makes its formal implementation a very hard task. A strategy to work around this issue is represented by the concept of **collaborative decision making** (**CDM**). The idea is to handle *equity*, directly involving airlines in the slot allocation process, trying to develop flexible mechanisms to provide them tools to express their needs and preferences.
In literature, one of the most important attempt to enhance the slot allocation procedure under a *CDM* framework, is the so called **at most-at least (AMAL)** model, proposed by Vossen and Ball [2006b]. The idea is to include the possibility to apply operations of type I and T, only if they satisfy certain constraints defined by the AUs themselves. The model is based on the following:

Assumption 3 *airlines might be interested in accepting a delay increase on some of their less important flights if they can obtain in return a reduction for their most strategical ones.*

In the formulation, this assumption is expressed introducing the concept of *two for two offers*. Given an airline A, a **two for two offer** (for brevity *offer* in this chapter) is a tuple $(f_d, s_{d'}; f_u, s_{u'})$ where f_u, f_d are flights of A and $s_{d'}, s_{u'}$ are slots within the hotspot. The offer $(f_d, s_{d'}; f_u, s_{u'})$ states that A is willing to allow an increase of delay for flight f_d, moving it down in the schedule to a slot no later than $s_{d'}$ (*at-most*), in return of a delay reduction for f_u, moving it up to a slot that is no later than $s_{u'}$ (*at-least*). Clearly the model requires also that:

Assumption 4 *airlines make offers that are convenient for them, which means that any exchange of slots, consequence of one of their offer, will produce a positive impact on them.*

To proceed with the model formulation we have to define:

- F, the set of all flights
- S, the set of all slots under regulation
- $\mathbb{A}$, the set of all airlines
- T, the set of all offers

We also use the following notation:

- f_i, that represents a specific flight. If f_i belongs to the airline $A \in \mathbb{A}$, the ownership will be indicated by $f_i \in A$
- the index i of f_i indicates the fact that f_i is the $i-th$ flight in the regulation flight list and that it is currently assigned to the $i-th$ slot $s_i \in S$

Suppose now that, according to the assumptions, all airlines made their offers. Chosen an airline A, let us consider a particular flight $f_i \in A$ the set of slots $\mathcal{S}_i$ corresponding to all specified *at-most* and *at-least* slots for f_i. $\mathcal{S}_i$ can be seen as a sequence of classes, which can be represented by the indices of the corresponding slots. The introduction of these classes might appear an additional complication, but we will show how it turns out to be a handy trick to ease the model formulation.
If for f_i, k_i offers have been made, we then have $|\mathcal{S}_i| = k_i$ with $\mathcal{S}_i = \{i_1, .., i_{k_i}\}$ and where we can assume that $i_1, .., i_{k_i}$ are increasingly ordered. Now, if there exists an index $j < k_i$ s.t. $i_j < i$ it means that there is an offer that tries to move up f_i; if $i < i_j$ instead, it means that there is an offer that tries to move f_i down. We have also to include the possibility that f_i is reassigned simply to its original slot s_i, this because of two reasons: 1) airline A does not define any offer for f_i, so $\mathcal{S}_i$ is empty; 2) no offers in $\mathcal{S}_i$ takes place. Toward this purpose we define for each flight a default offer: the tuple $(f_i, s_i; f_i, s_i)$ which has to be added to the set S_i. Notice now that, for each flight f_i, the set $\mathcal{S}_i$ cannot be anymore empty as it contains at least the default offer. Figure 3.3 shows the structure of the variables.

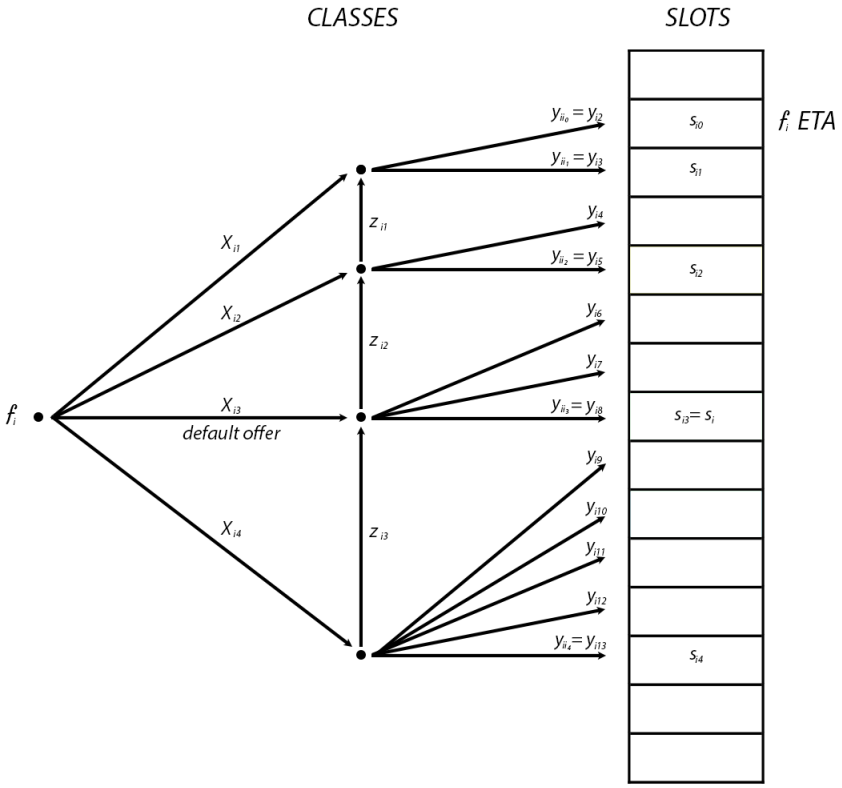


Figure 3.3: Variables structure

What we just defined is a mathematical framework to describe offers. Our goal, is now to develop a slot allocation model which is based on these offers. Let:

- $x_{ik} \in \{0,1\}$ for $f_i \in \mathcal{F}$, $1 \leq k \leq k_i$ be the decision variable s.t. $x_{ik} = 1$ *iff* f_i is assigned to class i_k

We have:

Constraint 2

$$\sum_{k=1}^{k_i} x_{ik} = 1 \quad \forall f_i \in \mathcal{F} \tag{3.2}$$

which indicates that exactly one offer has to be selected for flight f_i; it is important to remark that the default offer is included. Once a flight is assigned to an offer,then it has also to be assigned to a slot; let:

- $y_{ij} \in \{0,1\}$ for $f_i \in \mathcal{F}$, $s_j \in S$, s.t. $y_{ij} = 1$ *iff* flight f_i is assigned to slot s_j

Constraint 3

$$\sum_{f_i \in \mathcal{F}:\ i_0 \le j \le i_k} y_{ij} = 1 \quad \forall s_j \in S \tag{3.3}$$

More in detail, we want that if $x_{ik} = 1$, which means that offer i_k has been accepted, f_i has to be assigned to a slot earlier or equal to s_k. To achieve so, we have to introduce a slack decision variables:

- $z_{ik} \in 0, 1$ for $f_i \in \mathcal{F}$, $1 \le k \le k_i$,

whose role will be explained after the definition of the

Constraint 4

$$x_{i1} + z_{i1} - \sum_{j=i_0}^{i_1} y_{ij} = 0 \quad \forall f_i \in \mathcal{F}, \tag{3.4}$$

and the

Constraint 5

$$x_{ik} + z_{ik} - z_{ik-1} + \sum_{j=i_{k-1}+1}^{i_k} y_{ij} = 0 \quad \forall f_i \in \mathcal{F}, 1 < k < k_i \tag{3.5}$$

To understand how constraint 4, 5 and the variables z_{ik} work, and how they guarantee that if an offer is chosen then the flight is assigned to a slot earlier or equal to s_k, we have to proceed by cases. Fixed a flight f_i let us temporary suppress the index i to simplify the notation, implicitly implying that with x_k, we refer to x_{ik}. We also have to keep in mind that all flights have to be assigned, so in our case, this means $x_{ik} = x_k = 1$ for some k:

Case 1. $x_k = 1$ for $k = 1$

- 1) $\implies x_{k'} = 0 \quad \forall k' > 1$
- 3), 4) $\implies z_1 = 0$ otherwise we would violate 3), as $1+1-\sum_{j=i_0}^{i_1} y_j = 0$ and $\sum_{j=i_0}^{i_1} y_j \le 1$

As a result $y_j = 1$ for some j in $i_0, ..., i_1$. This means that the if the first class is selected, the flight is assigned to one of the slots defined by the first class, i.e. if $x_1 = 1$ the slot is chosen between $i_0, ..., i_1$, as desired.

Case 2. $x_k = 1$ for $k > 1$

- 1) $\implies x_j = 0 \quad \forall j \neq k$
- 4) $\implies 1 + z_k - z_{k-1} - \sum_{j=i_{k-1}+1}^{i_k} y_j = 0$

For simplicity, let us call $Y_k := \sum_{j=i_{k-1}+1}^{i_k} y_j$, where now $Y_k = 1$ means that the flight is assigned somewhere after slot i_{k-1} and before or at slot i_k. This allows us to rewrite the last equation as:

$$1 + z_k - z_{k-1} - Y_k = 0$$

which holds in one of the following sub cases:

Subcase (1): $Y_k = 1$ We are okay, since $y_j = 1$ for some j, $i_{k-1} < j \leq i_k$, which means that f_i is certainly assigned to a slot earlier than s_k, as desired.

Subcase (2): $Y_k = 0$

- 4) $\implies 1 + z_k - z_{k-1} - 0 = 0$
- 3), 4) $\implies$ $z_k = 0$ otherwise we would violate 3), as $1 + 1 - z_{k-1} - 0 = 0$ and $z_{k-1} \leq 1$
- $\implies z_{k-1} = 1$
- $\implies 0 + 1 - z_{k-2} - Y_{k-1} = 0$ for 4).
- $\implies z_{k-2} = 1 - Y_{k-1}$

  Subsubcase (1): $z_{k-2} = 0$ and $Y_{k-1} = 1$
  We are okay, as $y_j = 1$ for some j, $i_{k-2} < j \leq i_{k-1}$, which means that f_i is certainly assigned to a slot earlier than s_k, as desired.

  Subsubcase (2): $z_{k-2} = 1$ and $Y_{k-1} = 0$
  We can apply the same reasoning backward until we find an index j s.t. $Y_j = 1$ or index $k = 2$. In that case we would have $z_1 = 1$ and:

  - 3) $\implies 0 + z_1 - Y_1 = 0 \implies Y_1 = 1$

  and again we are okay as $y_j = 1$ for some j, $i_0 \leq j \leq i_1$, which means that f_i is certainly assigned to a slot earlier than s_k, as desired.

Another important **remark**: if the default offer is the one selected for a certain flight f_i, it does not necessarily imply that f_i is assigned to its original slot. In fact, the default offer states that f_i has to be assigned *at-most* (or *ar-least*) to its original slot. But this means that f_i can be also assigned to an earlier slot. This circumstance can happen (as shown in the following example) to allow other offers to take place. In other words, the default offer implies no change or a positive impact for f_i and this shows how the $AMAL$ model allows also operations of type $\mathcal{C}$.

The last condition we have to ensure, is that any slot trade has to be the resulting match of some offers, either proposed by the airlines or the default one. To achieve so we have to introduce:

- $D_T = \cup_{(f_d, s_{d'}; f_u, s_{u'})}(f_d, s_{d'})$ the set of classes corresponding to downward moves of all flights

- $U_T = \cup_{(f_d,s_{d'};f_u,s_{u'})}(f_u, s_{u'})$ the set of classes corresponding to upward moves of all flights
- $o_{dd',uu'} \in 0,1$ for $(f_d, s_{d'}; f_u, s_{u'}) \in T$ the decision variable s.t. $o_{dd',uu'} = 1$ if offer $(f_d, s_{d'}; f_u, s_{u'})$ has been selected

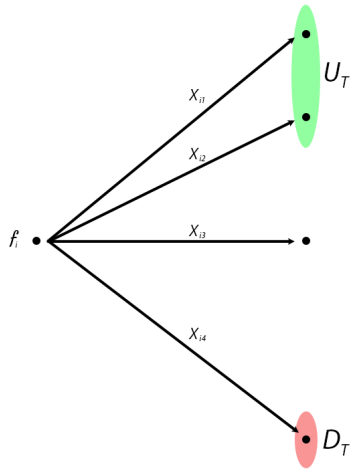


Figure 3.4: Representation of sets D_T and U_T

and define

Constraint 6

$$x_{dd'} = \sum_{(f_d,s_{d'};f_u,s_{u'})\in T} o_{(dd',uu')} \quad (f_d, s_{d'}) \in D_t$$

Constraint 7

$$x_{uu'} = \sum_{(f_d,s_{d'};f_u,s_{u'})\in T} o_{(dd',uu')} \quad (f_u, s_{u'}) \in U_t$$

To understand how it works, let us consider flight f_i and suppose that index k corresponds to the class of the default offer; we have:

Case 1. the default offer is chosen, so $x_{ik} = 1$
Constraint 2 guarantees that $x_{ik'} = 0$ for all $k' \in \mathcal{C}_i$ s.t. $k' \neq k$. In particular $x_{id'} = 0$ for all $(f_i, s_{d'}) \in D_T$ and $x_{iu'} = 0$ for all $(f_i, s_{u'}) \in U_T$; applying constraints 6 and 7 we obtain that, if the default offer is chosen for f_i, no other offers including f_i can be selected.

Case 2. an upper move is selected for f_i, i.e. $x_{iu} = 1$ for some $u < k$
Due to constraint 2 the default offer can't be anymore selected. Constraint 7

ensures that an offer including f_i is selected, i.e. $o_{(dd',iu)} = 1$ for some $(f_d, s_{d'})$. Constraint 6 then implies that:

$$1 = \sum_{(f_d, s_{d'}; f_u, s_{u'}) \in T} o_{(dd',uu')} = x_{dd'}$$

In other words, the flights f_i and f_d corresponding to the selected offer $o_{(dd',iu)}$ are assigned to the requested classes.

Case 3. a lower move is selected for f_i, i.e. $x_{id} = 1$ for some $d > k$
This case is analogous to the previous one.

3.2.1 Considerations

This model has been tested with several different objective functions, including an equivalent version of the one described in definition 11 and, as shown in Vossen and Ball [2006b], can lead to some significant improvement. Still, if a cost based objective is applied, the bias toward high cost flights remains a strong limitation for *equity* concerns. Furthermore, if on one hand the extra flexibility introduced with the offer mechanisms represents a remarkable improvement from the CDM prospective, on the other, the mechanism reveals some limitations under the practical point of view. In fact, each airline can make offers just based on preferences over its flights, but without having any information on what is available on the market. This means that an airline can define an offer which can be either infeasible or underestimating what it could actually achieve. The next example shows the two different scenarios:

Example 4 *Let us consider the situation depicted in figure 3.5, where airlines A and B respectively, formulate their offers for flights* $A1, A2, B1$ *and* $B2$ *in the way indicated by the arrows. For all other flights no offers have been made, i.e. it will be considered just the default offer.*

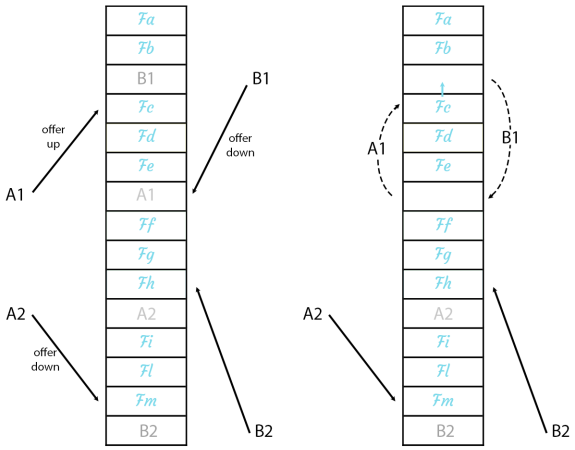
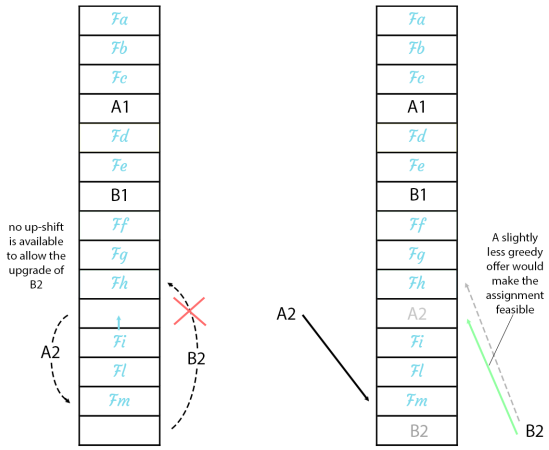


Figure 3.5: An example of offers

Flight $A1$ *can be assigned to its at-least slot, as airline* B *is willing to release the slot occupied by* $B1$ *and flight* $\mathcal{F}3$ *can be up shifted of one position. Similarly,* $B1$ *can be assigned to its at most slot. Unfortunately,* $B2$ *cannot be assigned to its at-least slot, as there is no slot available.*



It is important to notice that this fact implies that the corresponding offer cannot be chosen, and this does not allow the assignment also of $A1$ *and* $B1$*. Consequently, default offers will be considered for flights* $B1$ *and* $B2$*, which will cause the assignment to their original slot. So,* $B1$ *slot will be not released and this will also imply the offer regarding* $A1, A2$ *to result infeasible. The green arrow in the last figure is showing a slightly less "greedy" at-least offer for flight* $B2$*, that would provide a feasible solution.*
In figure 3.6 we see another possible scenario. This time offers are compatible and consequently an assignment that respects the airlines conditions, is possible.

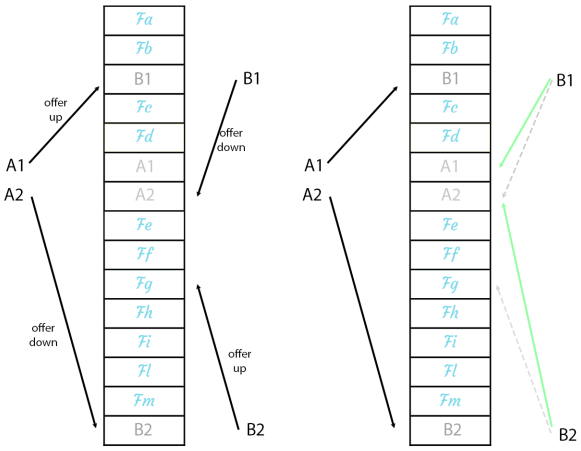


Figure 3.6: A feasible match of offers

However, airline B offer represents an underestimation of what it could actually achieve. In fact, the green arrows show a different offer formulation for flights B1 and B2, which would still provide a feasible solution, but more convenient. We remark that the the same solution could have been obtained even with the original offer, because the time constraints "at-least/at-most" are just lower bound conditions for the slot assignment, but in general this cannot be guaranteed or known a-priori.

These two examples show how the same offer can either be "too greedy" or "too conservative", just depending on the offers made by other airlines. In general, to formulate the offers in a feasible and possibly optimal way, one should have a at disposal a picture of what is available on the market, information very unlikely to obtain in a practical scenario.

3.3 A graph interpretation

We will see now a slightly different approach proposed by Sherali et al. [2011]. The model describes a framework very similar to the one of Vossen and Ball: the difference is that for each flight in an offer, rather then a *at-least/at-most* slot, an entire slot window is specified. It is easy to see that the *at-least/at-most* architecture can be still reproduced also in this model, just defining a window between the *ETA* and the *at-least/at-most* slots for flights included in some offers and the window between the *ETA* and the original slots for all the others. This model has been tested with different objective functions, obtaining several interesting results, but, similarly to the Vossen and Ball case, the same considerations about its practical limitations can be made. However, what we want to remark here, is the powerful graph interpretation of the slot trading

dynamics provided in the article. This approach in fact, will turn out very useful toward the formulation of our model, explained in chapter 6.

Let us examine the scenario depicted in the following figure 3.7:

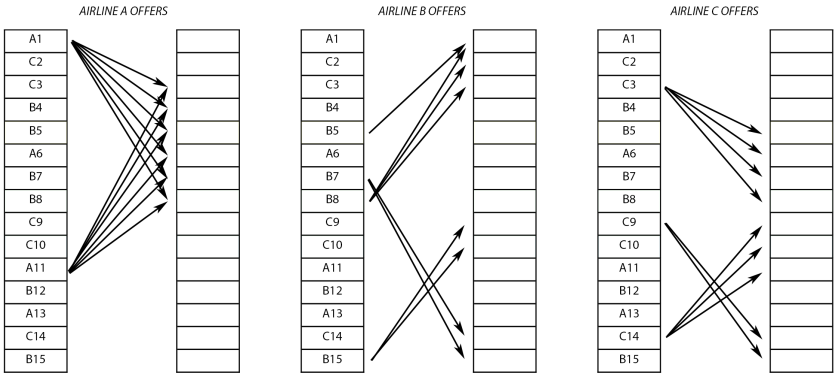


Figure 3.7: Offers representation

If we now consider a slot as node and the connection between a flight's current slot and all target slots as directed arcs, the whole slot trade configuration can be interpreted as a directed graph.

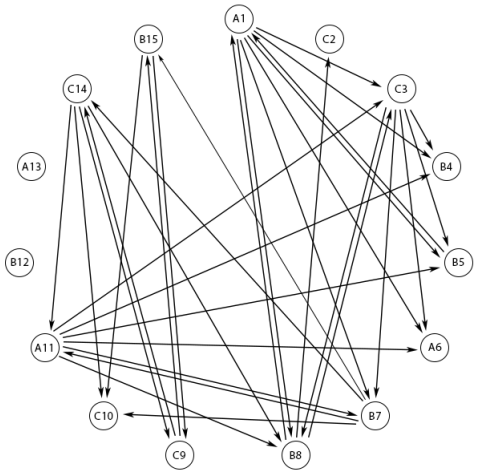


Figure 3.8: Graph representation of the slot trading offer mechanism

Consequently, all acceptable trades are represented by the directed cycles in graph.

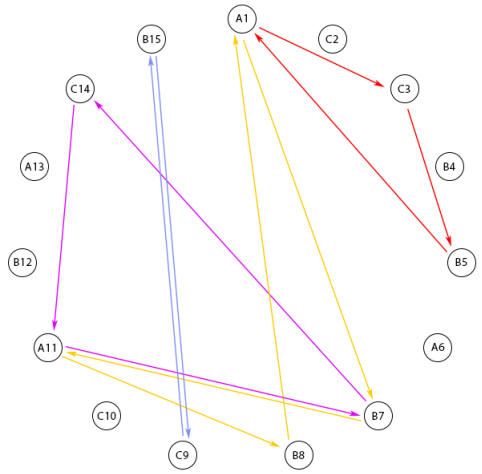


Figure 3.9: Some example of acceptable offers

Chapter 4

UDPP

4.1 Overview

The idea behind the UDPP mechanism is to improve flexibility and reduce delay impact on airlines under a CCS, allowing all airlines to perform operations of type $\mathcal{I}$ and type $\mathcal{C}$. In this chapter we will analyse in detail the UDPP algorithm defined in the document **SESAR Solution PJ07.02 SPR-INTEROP/OSED for V2 - Part I** SESAR [2019]. The features used in this UDPP version are:

Priority values: an airline can define for each of its flights a label that describes its level of importance. They represent the core of the UDPP concept as they are the values used by the algorithm to decide how to perform the operations.

Margins: a feature that allows airlines to additionally specify for any of its flights a target time window that will be considered by the algorithm for the slot assignment.

SFP ***Selective flight protection:*** an operations of type $\mathcal{C}$, or in some cases of type $\mathcal{I}$, that allows an airline to prioritise some of their most important flights in order to reduce as much as possible their delay. A flight chosen to be protected is called **Pflight** (protected flight). In exchange, for each *pflight* the airline must accept to increase the delay of one of its earlier flights.

FDR ***Flight delay reordering:*** an operation of type $\mathcal{I}$ that allows an airline to reshuffle the order of its flights based on the priority values given.

It is important to remark that under a CCS an airline can freely decide whether to use UDPP or not. If an airline decides not to participate in UDDP, its flights will be rescheduled based on the FPFS algorithm, and UDPP will try not to cause negative effects on them, even though, as we will see, it cannot be

always guaranteed. On the other hand, as long as UDPP includes operations of type $\mathcal{C}$, they might experience also a positive impact, due to the application of the SFP by some other airlines.
The flow of the UDPP mechanism can be summarised as follows:

- Up to a few hours before the beginning of the regulation, each airline declares its will to participate in the UDPP and submits the list of flights defining their priorities, flights priority values and/or *margins*.
- The UDPP algorithm is then separately applied for each airline, providing a **local solution** using the **Local Flight Time Assignment** (*local UDPP*). This means that, for each airline the algorithm produces a new schedule taking into account just its priority values and applying the UDPP features just on its flights. The *local UDPP* is a sequential algorithm and its workflow can be described as follows:
  - manage *Pflights*
  - manage flights with *Margins*
  - manage *default baseline* flights; in the case in which an AU sets a default priority value to **B** the *default baseline* flights are flights with no given explicit priority
  - manage flights with defined priorities (but no margins)
  - manage *suspended* flights; less important flights, that the AU allows to postpone at the end of the Hotspot (also candidate for cancellation)
- Eventually, all *local solutions* are merged to produce the final schedule in a FPFS manner, based on their new *local solution* times

4.2 UDPP features

We will now analyse in detail the different features provided by UDPP, right after introducing the following:

Definition 13 *A **UDPP measure** is the set of flights involved in a Capacity Constraint Situation.*

Definition 14 *The **Hotspot Flight Earlier Schedule** is an airport parameter (common to all AUs) which gives the maximum early departure or arrival delay buffer allowed by the airport to manage flights. (e.g. 5min = 5 minutes before schedule is allowed)*

Definition 15 *The Baseline Time is the time assigned to a flight by the FPFS algorithm*

4.2.1 Priority values

As we have seen before, the entire UDPP concept is based on the fact that AUs are allowed to indicate their flights priorities in order to obtain a different schedule and so reduce the delay impact caused by a CCS. The *priority values* are the UDPP representation of the AUs priorities, and for each flight they can be set to:

- a **priority number**. Conventionally in reverse order w.r.t the importance of a flight; e.g. from 1 (highest priority) to 999 (lowest priority). The scale used is a custom parameter which can be defined by each AU
- **P** to protect a flight. If a flight's priority is set to **P**, the flight will be called **Pflight** and it will be considered a very important AU's flight. For each *Pflight* two parameters have to be specified:
  - the **Max Delay Protection**: an AU's parameter that gives the maximum delay acceptable for a Pflight according to its schedule time: (e.g. 5min or 10min).
  - the *Hotspot Flight Earlier Schedule*.
- **L**, the lowest priority flight of the AU: equivalent to set the highest value in the AU priority number scale.
- **B**, baseline priority: it specifies to keep the baseline delay of the flight as the current target delay.
- **S**, to suspend a flight: it specifies that the flight will no longer be in the UDPP measure. It will be up to the AU to take a decision concerning this flight (cancellation, diversion, rerouting ...). The flight becomes the least important flight of the UDPP measure, and it will be allocated after the last not suspended flights of the UDPP measure.
- instead of assigning to each flight a priority value, an AU can decide to define a **default priority value** (either a number or a letter) that will be automatically assigned to a flight when no explicit priority is given. An important case has to be remarked:
  - **dB** default Baseline: is the priority value assigned to flights with no explicit priority defined, in the particular case in which an AU sets the *default priority value* to **B**. These flights will be called **dBflights** and they will be managed in a specific stage of the *local UDPP*.

4.2.2 Margins

Margins is an additional and optional UDPP feature introduced to provide to the AUs the possibility to specify time constraints on certain flights. When defined, *Margins* will be taken into account to rearrange flights in accordance to the requested time constraints (if feasible).
Margins are defined by two values:

- **Time not After**: specifies a time by which the flight is requested not to be later than the value indicated.
- **Time not Before**: specifies the earliest time of arrival of a flight; it is in general defined as:

$$Time\ Not\ Before = ETA$$

Observation 1 *If Margins are defined for Pflights they overwrite the Max Delay Protection and the Hotspot Flight Earlier Schedule. In this particular case we would have:*

$$Time\ not\ Before = ETA\ -\ Hotspot\ Flight\ Earlier\ Schedule.$$

In addition, it is important to remark that flights with defined *Margins*, called **Mflights**, will be considered more important than all the other non *Pflights*, meaning that the algorithm will try to reallocate them based on their priority value and their *Margins*, after having managed the *Pflights* **but** before handling all the others.

4.2.3 Selective Flight Protection (SFP)

Flight proteciton is a feature of UDPP, similar to the compression mechanism, that allows airlines to request a delay reduction of some of their flights if they accept an increase of delay for some others. More in detail, if a flight priority is set to P it is also given a time window defined by the *Max Delay Protection* and the *Hotspot Flight Earlier Schedule*. This time window defines a set of contiguous slots, called **Pobjective**, starting from the slot corresponding to the time given by *ETA - Hotspot Flight Earlier Schedule* and the one corresponding to the time given by *ETA + Max Delay Protection*. The goal is to assign to the *Pflight* a slot within the *Pobjective*. In order to allow this operation and avoid negative effects on other AUs, the AU

- **must** have, a minimum of one slot within or earlier the *Pobjective* of the protected flight, and this slot has not to be currently occupied by another *Pflight* (of the same AU).

In case of multiple *Pflights* this condition is not sufficient. In fact in case of n *Pflights*, with $n > 1$:

- each *Pflight* **must** be matched with an AU slot within or earlier than its *Pobjective* that has not been already matched with any other *Pflight*.

The idea is that once for a *Pflight* the matching slot s is identified (if possible), then:

- if s is within the *Pobjective*, the Pflight is simply assigned to s. The flight previously in s will be assigned to a later slot, owned by the AU, in the next stages of the UDPP algorithm.

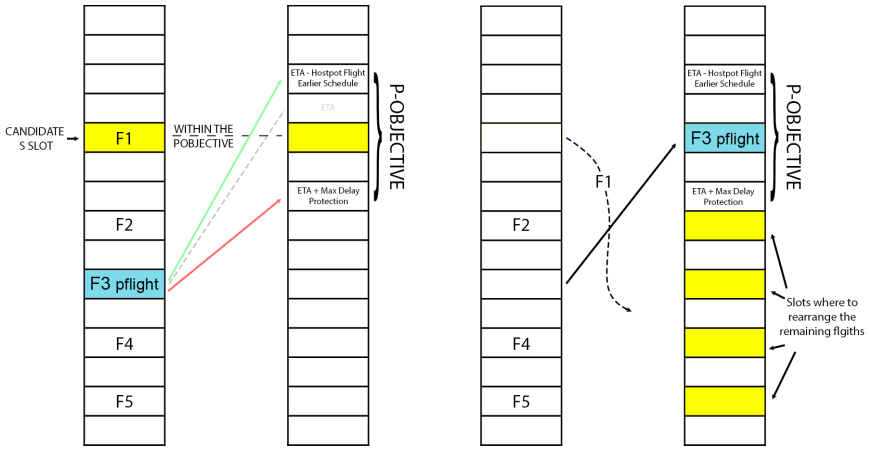


Figure 4.1: Protection when candidate slot s is in the *pobjective*

- if s is earlier than the *Pobjective*, then the *Pflight* is assigned to the slot corresponding to its *ETA*.

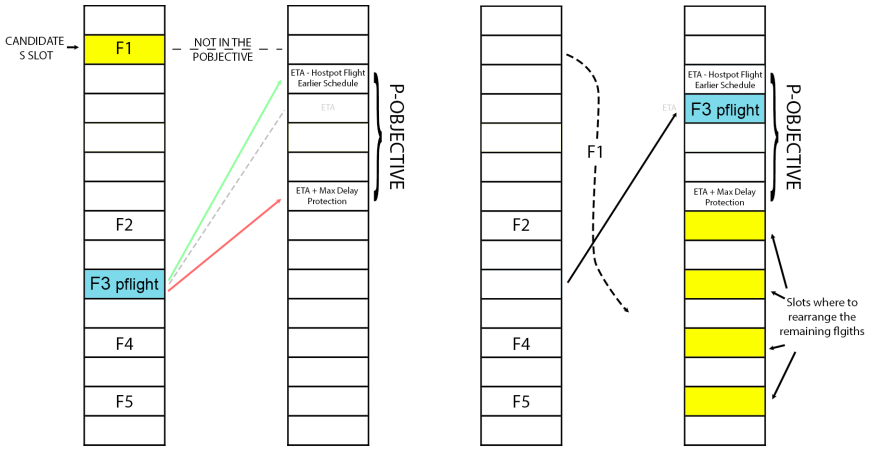


Figure 4.2: Protection when candidate slot s is not in the *pobjective*

The way in which a *Pflight* is matched with its target slot s will be discussed in detail later. For the moment we can still notice that this operation, is in the first case of type $\mathcal{I}$, as it does not effect other AUs. In the second case instead, we have that the *Pflight*'s new slot is currently held by another AU: we will see later how, in the final step, the *merge algorithm* menages this kind of circumstance in a $compression - like$ manner and how this might potentially

produce a positive effect on other AUs, defining an operation of type $\mathcal{C}$ (but also, in some quite rare cases, cause some negative impact).

4.2.4 Flight Delay Reordering (FDR)

This feature simply consists of a reallocation of the flights into the AUs' available slots which have not been already assigned by *SFP* to *Pflights*. The assignment is based on the priority values of the flights but also respecting the restrictions dictated by the *ETA*.
The algorithm sorts the flights by priority number and, starting from the highest priority flight to the lowest one, assigns to each flight the first remaining free slot compatible with its *ETA*.

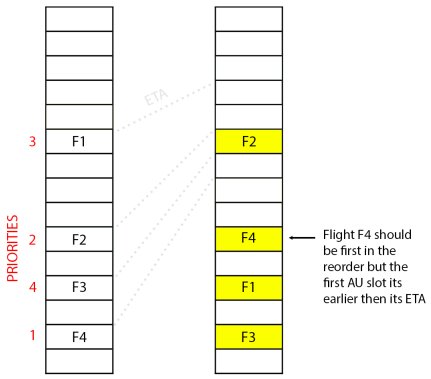


Figure 4.3: FDR

The same principle is beforehand applied to flights with defined *Margins*, but in this case just also considering the target time window specified by the *Time not Before* and *Time not After*.

4.3 Local UDPP

In this paragraph we will work in the UDPP local enviroment, so we will assume that just one AU is participating in UDPP and all UDPP features will be applied just to that airline. The schedule obtained in this manner will be the AU's *local solution*. This reasoning can be applied to all airlines actually participating to UDPP and later we will see how, starting from all *local solutions*, the merge algorithm will produce the final UDPP solution which will define the final schedule. In general terms the local UDPP can be seen just as a reordering of the flights, possibly within the slots owned by the AU. The flight protection, which is the first stage of the algorithm, is the only procedure that might assign a flight to a slot owned by another AU, but, as we have seen in 4.2.3, just if in exchange an earlier slot has been released.
We now show the definition of the *Local UDPP*:

Algorithm 3: Local UDPP

$slotList$ = set of slots owned by the AU without slots currently occupied by flights with priority B
$schedule$ = set of all slots
$Pflights$ = set of flights with priority value P
ManagePflights($Pflights, schedule, slotList$**)**

$Mflights$ = set of all *Mflights* which are not *Pflights*
$slotList$ = updated $slotList$ by *ManagePflights*
ManageMflights($Mflights, slotList$**)**

$dBFlights$ = set of all flights with priority value dB
ManageDBflights$(dBFLights, slotList)$

$Nflights$ = set of flights with priority number or default priority number but not *Margin*
ManageNflights$(Nflights, slotList)$

$Sflights$ = set of flights with priority S
ManageSflights$(Sflights, schedule)$

It is important to notice that in the first line of *Algorithm* 3 the initial $slotList$ does not include the $Bflights$. As long as all consequent functions will operate just on elements within the list, the $Bflights$ will remain untouched by the *Local UDPP*.

4.3.1 ManagePfLights()

The **ManagePfLights** consists in the **SFP algorithm** that we introduced in the section 4.2.3.
We remind that the inputs of this function are:

- $Pflights$: the list of flights with priority P
- *slotList*: set of slots owned by the AU without slots currently occupied by flights with priority B
- *shedule*: the list of all slots within the hotspot

The goal of this function is to assign, whether possible, all $Pflights$ to a slot within their *Pobjective* time window. We also remind that when *Margins* are defined, the *Pobjective* will be defined according to the *Time not After* and the *Max Delay Protection* will be ignored. Due to this fact we can then define and use simply the *Time not After* and the *Time not Before* in all cases, and when $Margins$ are not given just set:

$$\textit{Time not After} = \textit{ETA} + \textit{Max Delay Protection}$$
$$\textit{Time not Before} = \textit{ETA} - \textit{Hotspot Flight Earlier Schedule}$$

The algorithm can be then summarised in this way:

- it sorts the *Pflights* by **Time not After**
- sequentially, for each *Pflight*:
  - it tries to find in the *slotList* a slot s within the $Pobjective$.
    - if found, it assigns the $Pflight$ to s. s is then removed from the *slotList*
    - otherwise it assigns the $Pflight$ to the slot corresponding to the *ETA*. In this case the slot allocated to the $Pflight$ does not belong to the AU. In exchange, the last slot in the *slotList* earlier then the *Time not Before* is removed from the list. See figures 4.1 and 4.2
- if after all assignments, the order of the $Pflights$ (according to their *Time not After*) is not respected, their slot allocation is reordered.

We will see later that in the case in which a $Pflight$ is assigned to a slot of another AU, when the $UDPPmerge$ algorithm is executed, this might produce a positive effect on other AUs.
In the next detailed description we will refer to the *Time not After* as tnA and to the *Time not Before* as tnB.

Algorithm 4: ManagePfLights (SFP)

```
Pflights.sort() by tnA; if equal by Baseline Time

for pf in Pflights do
    solutionFound = false
    pfSlot = slot currently occupied by pf
    currentSlot = pf.tnAslot

    while solutionFound == false and currentSlot.time ≥ pf.tnB
     do
        if currentSlot in slotList then
            assign pf to currentSlot
            slotList.remove(currentSlot)
            slotList.append(pfSlot)
            slotList.sort() by time
            solutionFound = true
        end

        currentSlot = currentSlot − 1
    end

    if solutionFound == false then
        currentSlot = latest slot in slotList s.t. slot.time < pf.tnB
        pfNewSlot = slot in the schedule corresponding to pf.ETA
        assign pf to pfNewSlot
        slotList.remove(currentSlot)
        slotList.append(pfSlot)
        slotList.sort() by time
    end
end

if Pflights not sorted by Time not After then
    rearrange the Pflights slots order by Time not After; if equal by
      Baseline Time
end
```

4.3.2 The function *ManageTimeSolution*

The function ***ManageTimeSolution*** plays a crucial role in the management of all **non** *pflights*. The aim of this function is to assign a flight to a slot owned by the same AU as close as possible to a given target time; reminding that the *slotList* is the list of slots owned by the AU without the ones currently occupied by flights with priority B, we can summarise the *ManageTimeSolution* assignment procedure as follow:

- given a *target time* the function $getTargetSlot$ identifies the corresponding *target slot*, which is the latest one in the AU's *slotList* such that $slot.time \leq targetTime$.
- if the *target slot*
  - is free, the flight is assigned to it
  - is already assigned to another AU's flight, then
    - if some earlier AU's slot s is available it shifts earlier some flights between s and the *target slot*. This procedure, handled via the subfunctions *ManageSolutionEarlier* and *MoveFlightEarlier*, generates a free slot between s and the *target slot* and the flight will be assigned to it
    - if no earlier AU's slot are available (*ManageTimeSolution* returns *null*) the flight is assigned to the first later free AU's slot

An important **remark**: in the following pseudo code descriptions the slots will be represented by object variables which will hold all the information required, such as their status (free or used), their indexes, the flights assigned to them, etc. However, they might also assume the values *null* or *shiftBlocked*, meaning that in both cases no solution has been found, but for different reasons. *null* will indicate the fact that when looping backward through the *slotList*, the initial bound of the list has been excedeed. Instead, *shiftBlocked* will mean that a shift is not possible due to fact that a flight is blocked because of its ETA.

Algorithm 5: getTargetSlot(*targetTime, slotList*)

```
for i in range(slotList.length − 1) do
    if slotList[i].time ≤ targetTime and slotList[i + 1] > targetTime
     then
        return slotList[i]
    end
end
```

Algorithm 6 represents the pseudo code of the function *ManageTimeSolution*. Its task is to find a slot where to assign a given flight. Once identified the

targetSlot, the *solutionSlot* will be returned by the function *ManageSolutionEarlier*. In the only case in which the *targetSlot* and all earlier slots are already assigned, *ManageSolutionEarlier* will return the value *null* and the *solutionSlot* will be the first later free slot in the *slotList*.

Algorithm 6: ManageTimeSolution(*targetTime, flight, slotList*)

$targetSlot = getTargetSlot(targetTime, slotList)$
$solutionSlot = ManageSolutionEarlier(targetSlot, flight, slotList)$

if $solutionSlot == null$ **then**
  $solutionSlot = getFirstLaterFreeSlot(targetSlot, slotList)$
end

assign $flight$ to $solutionSlot$

The function *ManageTimeSolution* has been also designed to respect the hierarchy induced by the priorities: as long as the UDPP algorithm assigns all flights one after the other in order of priority, when *ManageTimeSolution* is handling the assignment of a particular flight, all already assigned flights are then considered more important. In this respect, in order to preserve the hierarchy, all flights which have already been assigned will, where possible, be shifted earlier. To better understand this procedure a detailed discussion of the *ManageSolutionEarlier* and the *MoveFlightEarlier* functions is needed.

The function *ManageSolutionEarlier*

At this stage we have to analyse how the function *ManageSolutionEarlier* identifies the *soutionSlot*. As we have mentioned earlier, the first step consists of calling the function *AvailableFreeSlotAtOrEarlier* to check whether the *targetSlot* is free or if there is an earlier available one. If this is not the case, *ManageSolutionEarlier* returns the *null* value and *ManageTimeSolution* will find a later slot for the flight. Alternatively, the function *MoveFlightEarlier* will try to identify a slot in the *slotList* between *slotList*[0] and the *targetSlot*, if necessary shifting earlier some of the already assigned earlier flights. If this last procedure does not succeed *MoveFlightEarlier* will return the *null* value and the *for* loop will ensure to find at least a later compatible slot. It is important to remark that the way in which the function $getFirstLaterFreeSlot$ searches for a later slot is different to the one performed by the *for* loop. In fact, the *for* loop takes place just if a free earlier slot exists (because of the previous call of the function *AvailableFreeSlotAtOrEarlier*). Therefore, in this second case it will still be possible to shift earlier some flights, as explained in example 5.

Algorithm 7: ManageSolutionEarlier(*targetTime, flight, slotList*)

```
if AvailableFreeSlotAtOrEarlier(targetSlot, slotList) == false then
    return = null
end
else
    for currentSlot in slotList[targetSlot.index : end] do
        solutionSlot =
          MoveFlightEarlier(currentSlot, flight, slotList)
        if solutionSlot ≠ null then
            return solutionSlot
        end
    end
end
```

Example 5 *In this example the first slot is available, therefore the function will start the for loop. However, flight $\mathcal{F}$ cannot be assigned earlier than its targetSlot due to its ETA and flights A and B cannot shift earlier for the same reason. As a consequence (as explained in detail in subsection 4.3.2) the first call to MoveFlightEarlier will return the null value. The targetSlot (currentSlot in the algorithm sketch) will then be shifted later of one position. In this case, flight C can be shifted earlier and its slot released and returned. ManageTimeSolution will then eventually assign $\mathcal{F}$ to it.*

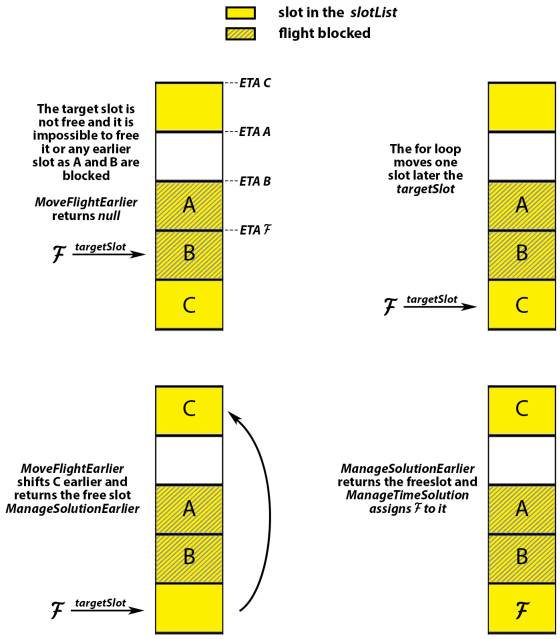


Figure 4.4: Example of how the *for* loop of *ManageSolutionEarlier* works

The function *MoveFlightEarlier*

Algorithm 8: MoveFlightEarlier(*targetSlot, flight, slotList*)

```
fromSlot = targetSlot
toSlot = slotList[targetSlot.index − 1]
while true do
    if flight.ETA > fromSlot.time then
        return shiftBlocked
    end

    if fromSlot.free == true then
        return fromSlot
    end

    if toSlot == null then
        return null
    end

    currentSlot =
      MoveFlightEarlier(toSlot, fromSlot.flight, slotList)

    if currentSlot ≠ null and currentSlot ≠ shiftBlocked then
        assign fromSlot.flight to the currentSlot
        fromSlot.free = true
        return from
    end

    if currentSlot == null then
        return null
    end

    fromSlot = slotList[fromSlot.index − 1]
    toSlot = slotList[fromSlot.index − 1]
end
```

The role of *MoveFlightEarlier* is to identify and return a slot where to assign the input flight. Its mechanism can be summarised as follows:

- if the *targetSlot* (*fromSlot*) is free, the function stops and returns the *targetSlot*
- alternatively, the flight currently assigned to the *targetSlot*, let's say f', has tried to be shifted earlier by a recursive call of the function itself, but with an earlier *targetSlot* and with f' as input flight.
  This procedure might lead to a sequence of recursive calls of *MoveFlightEarlier*. This circumstance occurs when, in order to free a slot, multiple flights

have to be shifted earlier.

- if a shift is not possible due to a blocked flight, let's call it bf (*id est* a flight for which its *targetSlot* it's earlier than its *ETA*), the *while* loop ensures that bf is ignored and the procedure continues with earlier flights (bf is skipped).

In order to better understand how this function works a few examples are required. For all the following instances we will assume that whenever a flight is non blocked (in which case the corresponding slot will be barred) there are no *ETA* restrictions, *id est* the condition

$$flight.ETA > fromSlot.free$$

will be always verified.

Example 6 *In this example the targetSlot for flight* $\mathcal{F}$ *is free. Therefore the condition*

$$fromSlot.free == true$$

is verified and the function simply returns the targetSlot.

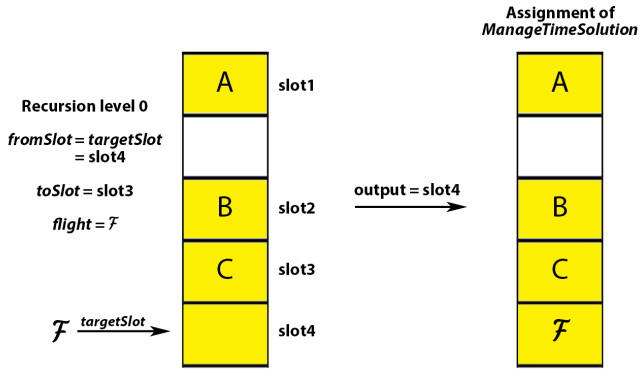


Figure 4.5: Example of how the *for* loop of *ManageSolutionEarlier* works

Example 7 *In the next example we have two recursion levels: level 0 and level 1. In the initial call of the function (level 0) the initial targetSlot for flight $\mathcal{F}$ is occupied by flight C. Consequently, a recursive call occurs (level 1):*

$$currentSlot = MoveFlightEarlier(slot3, C, slotList).$$

slot3 is free and returned. Back to level 0, we than have $currentSlot = slot3$. Eventually C is assigned to slot3 (C is shifted earlier) and slot4 is released and returned.

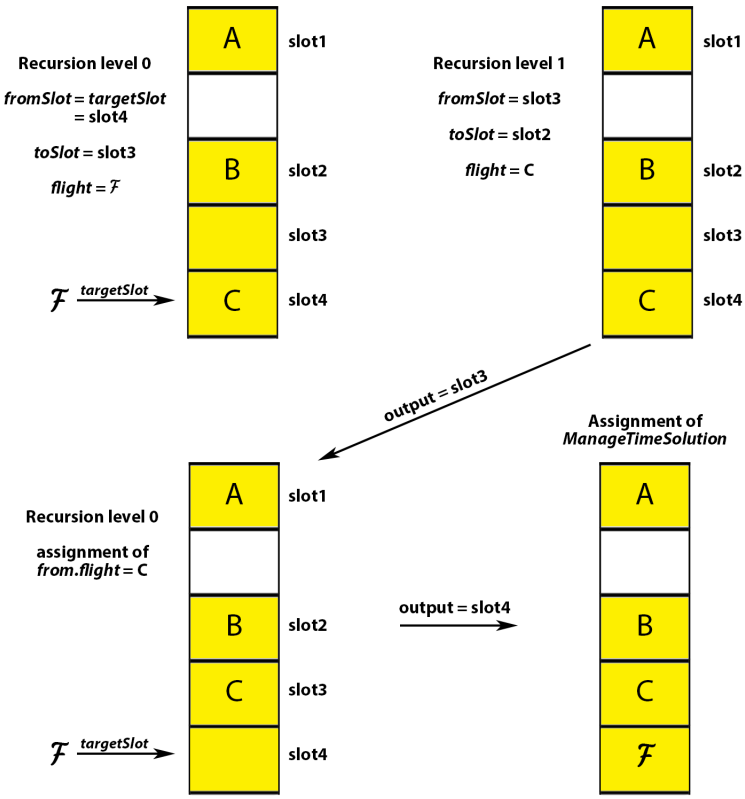


Figure 4.6: *targetSlot* free

Example 8 *In the next example, the initial targetSlot for flight $\mathcal{F}$ is once again occupied. But this time, in order to free slot4 both flights C and B have to be shifted earlier. This is achieved by three recursion levels of the function $MoveFlightEarlier$.*

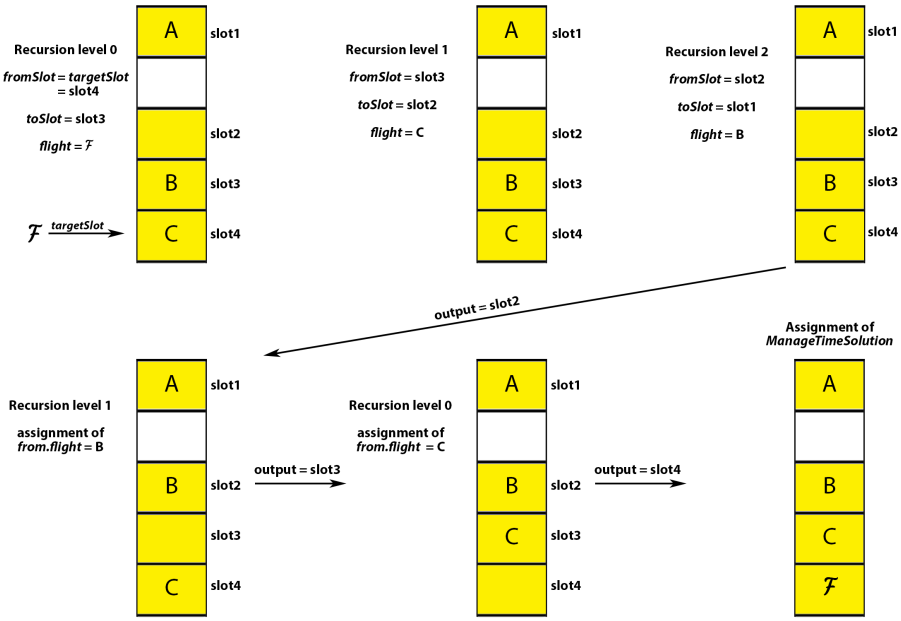


Figure 4.7: Flights C and B to be shifted, three recursion levels required

Example 9 *Similar to the previous example to free slot4 both flights C and B have to be shifted earlier, but in this case flight B is blocked. As a consequence at the third recursion level the condition*

$$B.ETA > slot2.time$$
$$(flight.ETA > fromSlot.time)$$

will not be verified and the value shiftBlocked will be returned. Back to the second recursion level, the fromSlot (slot2) will be shifted earlier in order to see whether it is possible an earlier assignment of flight C. A second iteration of the while loop will occur and the condition

$$C.ETA > slot2.time$$
$$(flight.ETA > fromSlot.time)$$

will be verified. With this example we are showing how during the shifting of earlier flights the initial order might be altered. More in detail, when a flight is blocked due to its ETA some later flight might overtake it and get an earlier position.

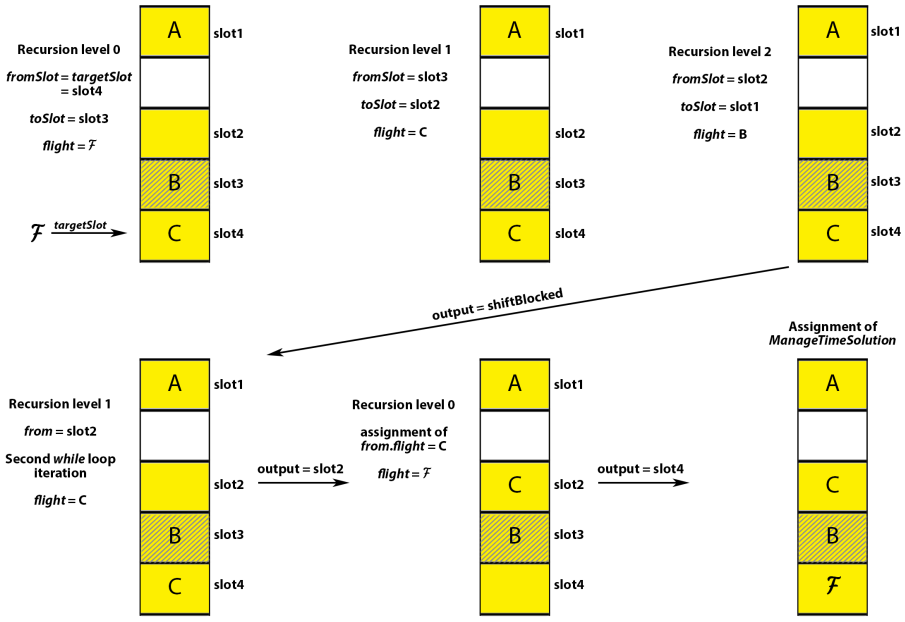


Figure 4.8: Example of how the *for* loop of *ManageSolutionEarlier* works

Example 10 *In this last example we want to show how it is possible to assign also to the initial target flight an earlier slot (compared to its initial targetSlot). In this case such circumstance occurs as both flights C and B are blocked and slot2 is then available for flight $\mathcal{F}$.*

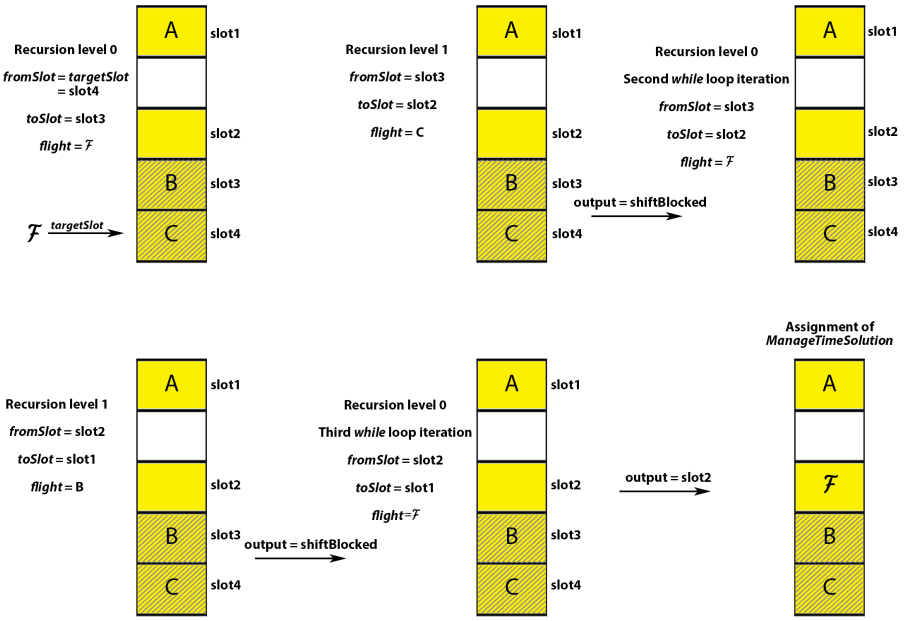


Figure 4.9: Example of an assignment of the initial target flight to an earlier slot compared to its initial *targetSlot*

4.3.3 ManageMflights

The inputs of this function are:

- *Mflights*: the list of the flights with defined *Margins* without the *Pflights*
- *slotList*, that we remind it has been updated by the function *ManagePflights*

The goal here, is to assign a slot to each *Mflight* respecting, where possible, its *Time not After* margin. The algorithm can be summarised in this way:

- it sorts the *Mflights* by priority number, if equal by *Time not After*, if equal by FPFS assigned time
- sequentially, for each $Mflight$:
  - the function *ManageTimeSolution* is called with *target time* equal to the *Time not After*

  Reminding that we are referring to the *Time not After* as tnA, the detailed description of the function can be given as follow:

Algorithm 9: ManageMflights($Mflights, slotList$)

$Mflights.sort()$ by priority number, if equal by tnA; if equal by FPFS assigned time

for mf *in* $Mflights$ **do**
  $ManageTimeSolution(mf.tnA, mf, slotlist)$
end

4.3.4 ManageDBflights

The objective of this function is to assign all flights with default priority B. The inputs of this function are:

- $dBflights$: the list of all flights with default priority B, which we remind is the default assignment given to a flight when no explicit priority is given to it
- *slotList*: updated by the function *ManagePflights*, but including slots assigned by *ManageMflights*

The reason why the *slotList* still contains also the *Mflights* is that in order to handle the $dBflights$, the function *ManageTimeSolution* (*algorithm* 6) is used, so it might still be possible to shift earlier some $Mflights$.

Algorithm 10: ManageDBflights($dbFlights, slotList$)

$dBflights.sort()$ by Baseline Time;

for dbf *in* $dBflights$ **do**
    $targetTime = dbf.baselineTime$
    $ManageTimeSolution(targetTime, dbf, slotList)$
end

4.3.5 ManageNflights

The input of this functions are:

- $nFlights$: the list of the flights with priority number with no $Margins$ defined, and with no priority of type P, B or S.
- $slotList$: including slots assigned by $Mflights$ and $dBflights$
- $freeSlots$: the sublist of the $slotList$ containing just the free slots
- $hotspotEnd$: time of the last Hotspot slot

The idea, is to allocate all $nFlights$ on the remaining free slots according to their priority number. The algorithm sorts the flights by priority number and, starting from the highest priority flight to the lowest one, assigns to each flight the first remaining free slot. In case an assignment cannot be performed due to the flight ETA, the function $ManageTimeSolution$ is then called, with the target time set to end of the hotspot. The algorithm can be described in details as follow:

Algorithm 11: ManageNflights(nflights, slotList, freeSlots)

$nFlights.sort()$ by priority number, if equal by Baseline Time;
$freeSlots$ = slots in $slotList$ which are not assigned to any flights yet

for nf *in* $nFlights$ **do**
    if $freeSlots[0].time \geq nf.eta$ **then**
        assign nf to $freeSlots[0]$
        $freeSlots.remove(freeSlot[0])$
    else
        $ManageTimeSolution(hotspotEnd, nf, slotList)$
    end
end

4.3.6 ManageSflights

During this stage of the algorithm, to all the $sFlights$, that we recall are those that will no longer be in the middle of the UDPP measure as considered the least important flights of the UDPP measure, will be simply assigned the $hotspotEnd$ time. The actual slot allocation will be performed by the $UDPPmerge$ algorithm.

4.3.7 Local UDPP example

Let us now visualise the flow of the *UDPPlocal* with the following:

Example 11 *Here, we consider an AU with 5 flights: one protected, no suspended, two with Margins and two for which just priority numbers are given. For simplicity we suppose that for flight F3 the Hotspot Flight Earlier Schedule = 0, so that Time not Before = ETA.*

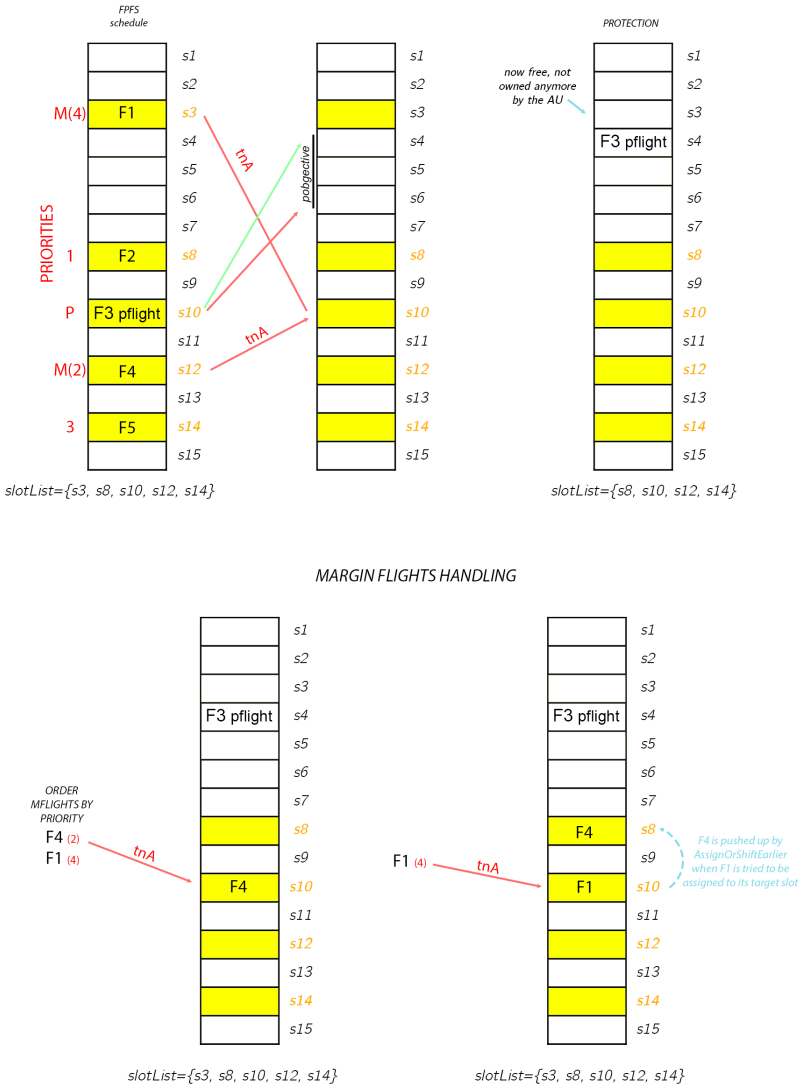


Figure 4.10: *UDPPlocal* example

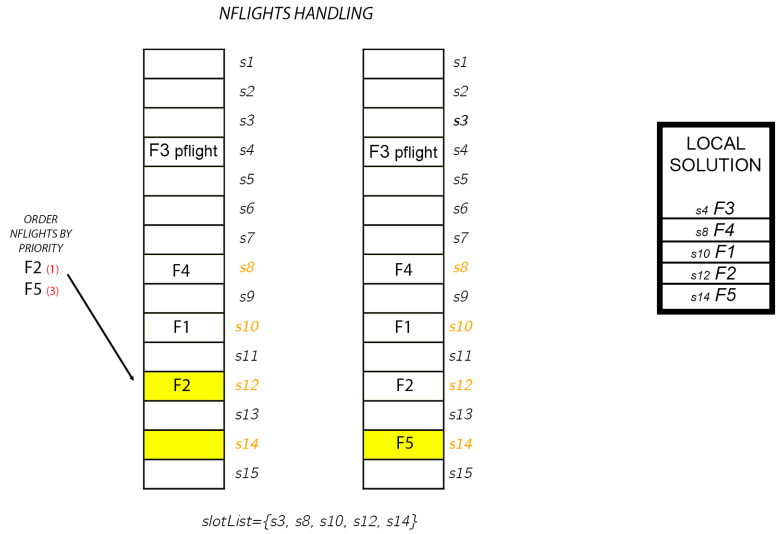


Figure 4.11: *UDPPlocal* example

4.4 UDPPmerge

This is the last step of the algorithm, in which the new final time schedule will be eventually computed. The idea is to try to assign all flights to the resources defined by all AUs' *localSolutions*. The local list of the AUs which are not participating to UDPP is by default simply defined by the one generated assigning to each flight its baseline slot.
The procedure can be summarised as follow:

- get all flights which are not *sFlights* from all AUs
- sort them according to time assignment by the *localSolutioin* and in case of time conflict according to the Baseline Time
- sequentially, for each flight:
  - assign it to the first available slot not earlier then the time given by

    ETA - Hotspot Flight Earlier Schedule.

    This requirement is crucial to avoid negative impact for later flights (as explained in the example 13)
- create an *emptyList* with the slots still unassigned within the hotspot, and append also empty slots available after the end of the hotspot
- get all *sFlights* and sort them by Baseline Time

- sequentially, for each *Sflight*:
  - try to fill the first slot in the *emptyList* that is later or equal to the flight ETA

Before showing the details of the algorithm we remind that if an AU did not participate in UDPP, for each of its flight we set $flight.newSlot = flight.oldSlot$.

Algorithm 12: UDPPmerge

$flights$ = list of all non $sFlights$ within the hotspot
$flight.sort()$ by their $newSlot$ parameter, the one obtained with the $UDPPlocal$, if equal by Baseline Time
$slotList$ = all slots in within the hotspot
$CASAlikeAssignment(flights, slotList)$

$sFlights$ = list of all $sFlights$ of all airlines
$sFlight.sort()$ by Baseline Time
$freeSlots$ = slots in $slotList$ which are not assigned to any flights yet
$laterFreeSlots$ = set of $size(sFlights)$ free slot available after the end of the hotspot
$freeSlots.append(laterFreeSlots)$
$CASAlikeAssignment(sFlights, freeSlots)$

Algorithm 13: CASAlikeAssignment($flightList, slotList$)

```
for f in flightList do
    flightAssigned = false

    while flightAssigned = false do
        i = 0

        if slotList[i].time >= f.ETA then
            assign f to slotList[i]
            slotList.remove(slotList[i])
            flightAssigned = true
        else
            i+ = 1
```

To summarise the flow of the *UDPPmerge* lets see an:

Example 12 *In this example we have an hotspot involving 4 airlines, a, B, c, D. Airlines a and c decide not to use the UDPP, so their local solution will be the one provided by FPFS. B and D instead, decide to participate in UDPP, and from UDPPlocal they obtain the following local solutions:*

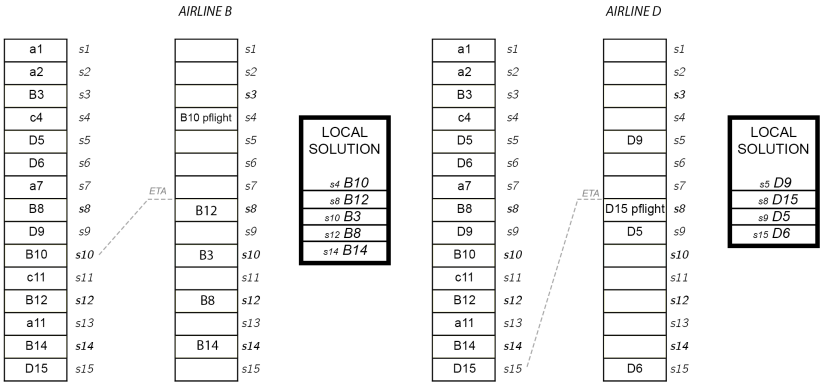


Figure 4.12: Local solutions for airline B and D

For simplicity we will once again assume that for all flights we have

$$Hotspot\ Flight\ Earlier\ Schedule = 0.$$

The UDPPmerge will first create the $flightList$ including all flights within the UDPP measure, sorted by local solution time and then (in case of conflict) by ETA. Then, respecting the new list order, the algorithm will assign each flight to the the first free slot; if for a certain flight the first free slot is earlier its ETA, the algorithm will assign it to the first compatible slot. In the example, this is the case of flight $B12$, which cannot be assigned to slot $s7$:

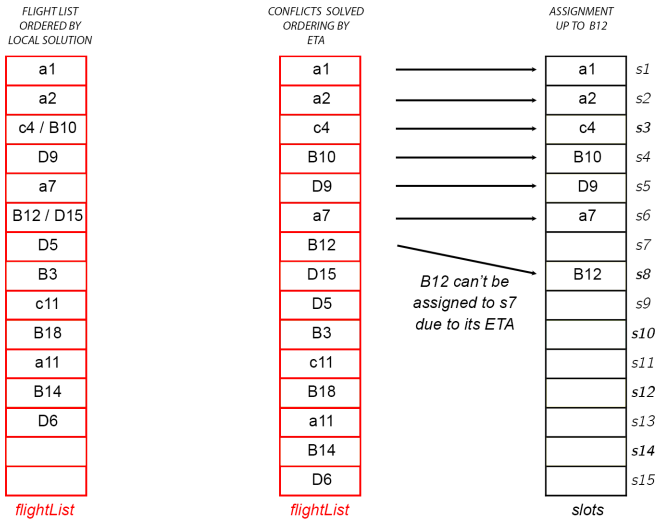


Figure 4.13: Assignment up to flight $B12$

$s7$ is then free, but next flight, $D15$, cannot be assigned to it as, again, it is

earlier than its ETA. s8 is occupied by B12, so D15 is assigned to s9. Then, s7 is still the first free slot, but this time it is compatible with next flight D5's ETA:

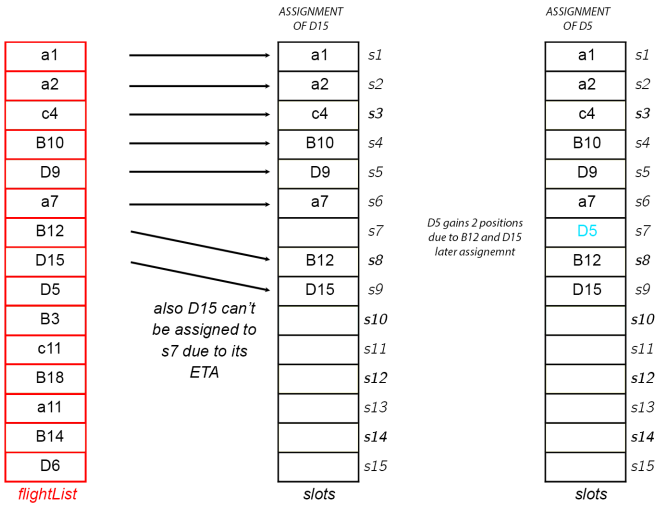


Figure 4.14: Assignment of flight $D15$

As from this stage, there are no other ETA conflicts, all other flights are smoothly assigned to the first free slot.

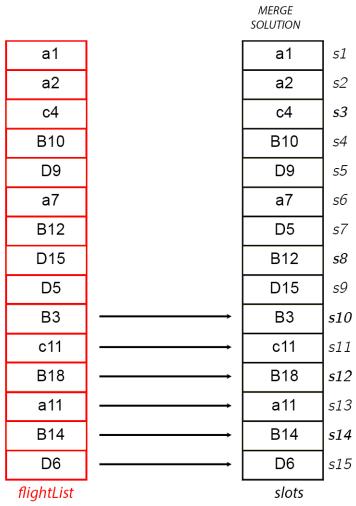


Figure 4.15: Last assignemts

4.5 Considerations

4.5.1 Positive impact

As we have seen in the previous example the flight protection can lead to a positive impact on other AUs, in fact flight $D5$ is gaining one position in the schedule and so reducing its delay. More in detail, this can happen when the slot released to apply protection is not in the *pobjective* of the flight protected, this defines an operation of type $\mathcal{C}$ and more in general it can produce a positive effects on multiple flights as we can see in the figure below.

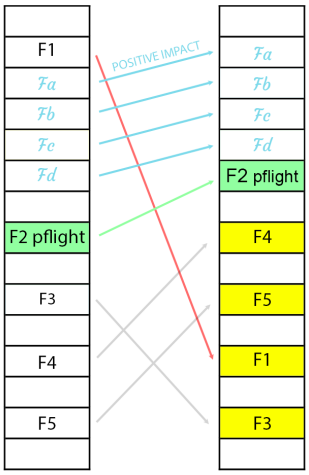


Figure 4.16: Positive impact caused by protection of flight $F2$

4.5.2 Delay deterioration

One of the main targets of UDPP is to reduce the delay impact, trying to guarantee that the schedule changes proposed in the local solution for each AU, do not produce undesired effects on other AUs after the merging stage. By "undesired" we mean that if an airline, for each of its flights, would be favourable to keep the same slot or receive an earlier one, compared to the one assigned by the local solution (if the AU participate in UDPP) or the FPFS (otherwise), it would not accept a later one.
Clearly, if all local solutions are obtained performing only operations of type $\mathcal{I}$, this will guarantee no delay deterioration for any AU. In fact in each local solution all flights will be rearranged just within the slots already owned by the same airline, as we can see in figure 4.17. For example, this is the case when only *FDR* operations are performed. The same consideration holds also for *SFP* operations in the case in which all the matching slots for all the *Pflights* are within the respective *Pobjectives*. In fact, as we have seen in section 4.2.3, in this case all the protection operations are of type $\mathcal{I}$. More complex is the

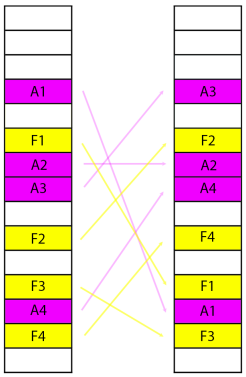


Figure 4.17: FDR applied by two airlines

situation in which operations of type C are performed, so when protecting a flight, its matching slot is not within the *Pobjective*.
Unfortunately, in this circumstance it might theoretically happen that some airlines experience a negative impact, as shown in the following:

Example 13 *Let us suppose to have the situation described in figure 4.18, where airlines A and B want to protect their flights A6 and B7 and consequently releasing the first two slots. Let us also assume that*

$$Hotspot\ Flight\ Earlier\ Schedule = 0,$$

so that the earliest time at which each Pflight can be assigned is simply represented by its ETA.

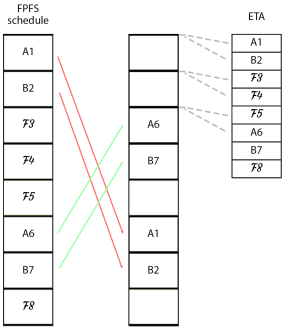


Figure 4.18: Protection applied by two airlines

The UDPPmerge will then generate the following flightList:

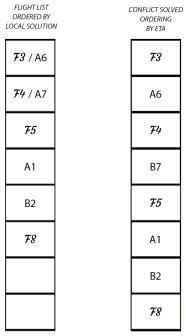


Figure 4.19: *UDPPmerge flightList*

When starting to assign the flights, the algorithm cannot give the first slot to $\mathcal{F}3$ as it is earlier then its ETA. Instead, it is assigned to the next one: slot 2.

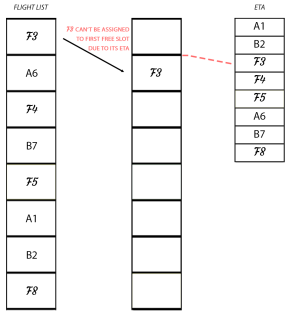


Figure 4.20: First assignemt

A6 cannot either take slot 1. But slot 2 is already assigned to $\mathcal{F}3$, so it has to be assigned to slot 3:

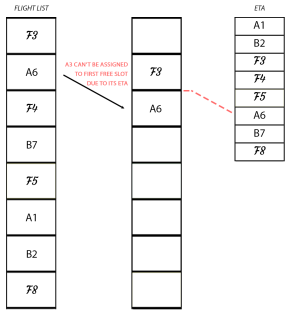


Figure 4.21: Second assignemt

The same issue occurs assigning the following flights, so in the end slot 1 remains free:

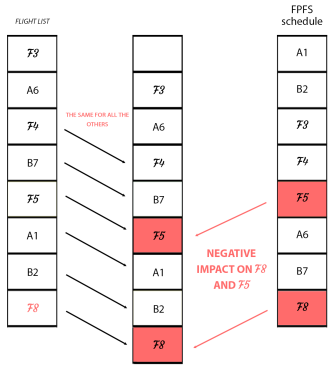


Figure 4.22: Final result

As a result, flights $\mathcal{F}3$ and $\mathcal{F}8$ are downgraded of one position, incurring in a delay deterioration.

Fortunately this last example represents a quite rare scenario, as in practice it turns to be unlikely to happen. The reason lies in the fact that the issue occurs when the *UDPPmerge* creates a hole in the schedule which cannot be filled by any consecutive flight, due to some ETA conditions. However, the fact that in practice the parameter *Hotspot Flight Earlier Schedule* is generally greater than zero guarantees enough flexibility to avoid this circumstance to occur. Moreover, in this example the issue arises as the *Pobjectives* of the *Pflights* are set around the earliest slots of the *UDPP measure* and this represents a very unlikely scenario in practice.

4.5.3 Local optimality

The main strength of UDPP is that it gives to the airlines a very flexible tool to reduce the delay impact, trying to find a better solution w.r.t. the FPFS one, also without an accurate knowledge of the costs function. This is due to the fact that if an AU has got at disposal the *UDPPlocal* algorithm, it can try, before officially submitting its priorities, to get from *UDPPlocal* an indicative picture of what its final schedule will look like after the *UDPPmerge* is applied. So, different *what if* scenarios can be obtained just testing different priority assignments, and this can be repeated until a satisfactory solution is found, determining in this way the AU's final priority settings.
But if we assume to have at disposal an accurate cost function, it can be shown that there exists a model to optimise the local solution, *id est* there is a priority assignment that guarantees a local optimum in terms of delay impact. The optimum is local in the sense that it is the best solution in terms of AU's local

schedule; as we have seen, the *UDPPmerge* might produce a slightly different final schedule. In order to prove it, we can first observe that if no protection or suspension is applied, any local solution will be simply determined by FDR operation, so a reordering of the flights within the AU's slots. In particular, it is easy to show if no protection or suspension is applied any local solution can be obtained just with the use of priority numbers. Let us suppose that $\mathcal{S}$ is a feasible local solution obtained using priority numbers, $Margins$, B priority value and dB as default priority value. Now, the same solution would have been achieved just using priority numbers if the values where given in ascending manner, according to the order defined by $\mathcal{S}$. In other words, if an airline, using all *UDPPfeatures* would have gotten the solution $f4, f2, f7, f1, \ldots$, the same result would been achieved using only priority numbers, if assigned in this way: 1 to $f4$, 2 to $f2$, 3 to $f7$, 4 to $f1$. In fact, $ManageNflights$ would have sorted the $flightList$ by priority number obtaining the same order.

If we now include the protection feature we have to distinguish two scenarios: if for a protected flight f_p, there exists an AU's slot within the $Pobjective$, this is again equivalent to an FDR operation; so, as before, we could have obtained the same solution using priority numbers only. Different is the situation where no AU's slot is within the $Pobjective$, as $ManageNflights$ cannot allocate any flight outside the AU's slots. But in this case, the protection would simply assign the $pflight$ to its ETA slot and redefine the AU's slot list. The new list will be obtained from the original one, just removing the closest earlier slot to the f_p's $Pobjective$. The others non $pflights$ would then be allocated into the new slot list in a FDR manner. In the end, we can then interpret $UDPPlocal$, simply as a combination of SFP and FDR. So far we did not consider the *suspension* feature, but for the purpose of our analysis we can assume that no airline intends to cancel or heavily penalise any of its flight and it is just interested in optimising its resources.

These considerations allow us to design an LP optimisation model to find the best feasible local solution achievable using SFP and FDR; toward this purpose we need to introduce:

- N, the number of slots within the UDPP measure
- A, the AU
- K the number of flights of A within the hotspot
- F, the set of the flights of A
- Index i will either indicate the flight i and the slot it currently occupies
- S the set of all slots, $S := \{1, ..., N\}$
- S_A, the set of indexes from 1 to K, that represent the slots owned by A, $S_A := \{1, ..., K\}$

- $\mathcal{IS}_A$, the set of indexes corresponding to the actual position in the schedule of the slots/flights owned by A
- $\mathcal{IS}_A(i)$, the function $\mathcal{IS} : S_A \to S$ that returns the position of slot/flight i in the schedule (in S)
- $x_{ik} \in \{0, 1\}$, flight originally assigned to slot i is assigned to slot $\mathcal{IS}_\mathcal{A}(k)$ (it describes the FDR allocation)
- $y_{ij} \in \{0, 1\}$, flight originally assigned to i is assigned to j (it describes the SFP allocation)
- $d_{ik} \in \mathbb{R}^+$, delay of flight i when assigned to slot k
- $\mathcal{C}(d_{ij})$, cost function, with $\mathcal{C} : \mathbb{R}^+ \to \mathbb{R}^+$

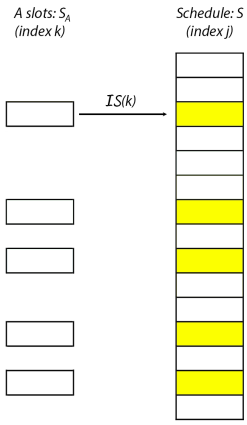


Figure 4.23: Variables meaning

We have now to define the following:

Constraint 8 *The first flight cannot be protected as no earlier flight can be downgraded in exchange. Therefore it must to be assigned to a slot owned by A:*

$$\sum_{k \in S_A} x_{1k} = 1$$

All others flights have to be either protected or assigned to a slot owned by A:

$$\sum_{k \in S_A} x_{ik} + \sum_{j \in S} y_{ij} = 1 \quad \forall i \in F - 1$$

Constraint 9 *All slots can host at most one flight:*

$$\sum_{i \in F} x_{ik} \leq 1 \quad \forall j \in S_A$$
$$\sum_{i \in F} y_{ij} \leq 1 \quad \forall j \in S$$

Constraint 10 *All flights cannot be assigned to a slot earlier than their respective ETA:*

$$\sum_{k < ETA(i)} x_{ik} = 0 \quad \forall i \in F$$
$$\sum_{j < ETA(i)} y_{ik} = 0 \quad \forall i \in F$$

Constraint 11 *Protect a flight means to allocate it to an earlier slot. Also, in this model, the slot requested for protection has not to be owned by A, otherwise, under our interpretation, this would be described by an FDR operation, so by some variable x_{ik}:*

$$\sum_{j > i} y_{ij} = 0 \quad \forall i \in F$$
$$\sum_{j \in \mathcal{IS}_A} y_{ij} = 0 \quad \forall i \in F$$

Constraint 12 *To guarantee that for each protection, a flight is downgraded in exchange, we have to impose that for each AU's slot k, the number of protected flights which where originally scheduled to k or after k, but assigned before k, must be lower than the number of flights moved from earlier than k to later than k. In other words, given k, the flow of $pflights$ going from lower than k to higher than k, must be lower or equal than the flow of non $pflights$ going from higher than k to lower than k :*

$$\sum_{i \geq k, j > k} y_{ij} \leq \sum_{i \leq k, t > k} x_{it} \quad \forall k \in S_A$$

Indeed, if a protection is applied, the closest earlier slot to the *pobjective* has to be released. This means that if $y_{ij} = 1$ for some j, and if k is the greatest index such that $\mathcal{IS}_{\mathcal{A}}(k) < j$ then we have $x_{ik} = 0 \; \forall i \in F$ (*id est* no flight can be assigned to the k-st slot owned by A, which is the $\mathcal{IS}(k)$-st in the schedule). To achieve this we have to introduce:

- $z_k \in \mathbb{N}$, the number of $pflights$ assigned between $\mathcal{IS}(k)$ and $\mathcal{IS}(k+1)$

Constraint 13 *formally:*

$$z_k = \sum_{i \in F,\ \mathcal{IS}(k) < j < \mathcal{IS}(k+1)} y_{ij} \quad \forall k \in S_A - \{K\}$$

Constraint 14 *The following inequality, guarantees that if $z_k > 0$, the k-th slot is released (no flights is assigned to it) and if $z_k > 1$, also the corresponding previous $z_k - 1$ slots of A:*

$$(1 - \sum_{i \in F} x_{ik'}) \cdot \mathcal{M} \geq z_k - (k - k') \quad \forall k \in S_A - \{K\},\ \forall k' \leq k$$

Where $\mathcal{M} >> 0$. Let us see how this constraint works: if a $pflight$ is assigned to a slot within $\mathcal{IS}(k)$ and $\mathcal{IS}(k+1)$, then $z_k = 1$ and no flight can be assigned to the $\mathcal{IS}(k)$-th slot (it has to be released). This implies $x_{ik} = 0; \forall i \in F$. This holds as otherwise we would have:

$$0 = (1-1) \cdot \mathcal{M} = (1 - \sum_{i \in F} x_{ik}) \cdot \mathcal{M} \geq z_k - (k - k) = z_k = 1$$

For all earlier AU' slots k', so s.t. $k' < k$ we have :

$$z_k - (k - k') = 1 - (k - k') \leq 0$$

so no restriction is made over the values of $x_{ik'}$ (the corresponding slot are still assignable).

If two $pflights$ are assigned within $\mathcal{IS}(k)$, and $\mathcal{IS}(k+1)$, then $z_k = 2$ and no flight can be assigned either to the $\mathcal{IS}(k)$-th and the $\mathcal{IS}(k-1)$-th slots. This implies $x_{ik} = 0$ and $x_{i(k-1)} = 0; \forall i \in F$. This holds again as otherwise we would have:

$$0 = (1-1) \cdot \mathcal{M} = (1 - \sum_{i \in F} x_{ik}) \cdot \mathcal{M} \geq z_k - (k - k) = z_k = 2$$

and

$$0 = (1-1) \cdot \mathcal{M} = (1 - \sum_{i \in F} x_{i(k-1)}) \cdot \mathcal{M} \geq z_k - (k - k - 1) = z_k = 1$$

For all earlier AU'slots k', so s.t. $k' < k - 1$ we again have:

$$1 = z_k - (k - k - 1) = 2 - (k - k - 1) > z_k - (k - k')$$

so:

$$z_k - (k - k') \leq 0$$

and again no restriction is made over the values of $x_{ik'}$. The same reasoning can be done for greater values of z_k.

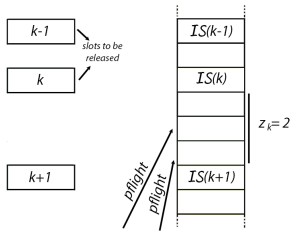


Figure 4.24: z_k meaning

Our objective is is to minimise the costs, so:

Definition 16

$$OBJ := \min(\sum_{i\in,k\in S_A} \mathcal{C}(d_{ik})x_{ik} + \sum_{i\in,j\in S} \mathcal{C}(d_{ij})y_{ij}) \tag{4.1}$$

This model has been used in our simulations to optimise the local solutions. Figure 4.25 shows a solution example obtained with our model. We remark that same result could have been obtained applying the *UDPPlocal* algorithm with an appropriate usage of flight protection and priority numbers.

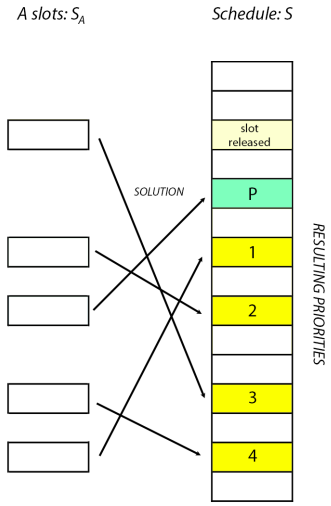


Figure 4.25: Example of correspondence between the model solution and the priorities

Chapter 5

The Inter-airlines slot offers provider model

5.1 Introduction

The model we propose in this chapter, named **Inter-airlines slot offers provider** (*ISTOP*), aims to extend the possibility of operations of type $\mathcal{T}$ within the $UDPP$ framework. We will show that such extension can potentially provide a further improvement in terms of total delay impact and equity: if on one hand the *UDPP* mechanism already ensures good results especially for airlines with a great number of flights, on the other, smaller AUs might not have enough flexibility to exploit the *UDPP* features, in order to improve their situation. Our idea is to use AUs preferences as equity measurement and enhance flexibility introducing new *CDM* tools. Before the introduction of the *ISTOP* model, a quick recap of the scenario in which we are operating is necessary. The assumption are:

- a CSS has been declared and it involves a set of flights F
- FPFS has been already applied
- *(optional) UDPP has been already applied*

The third hypothesis is optional in the sense that although the model has been designed to work in addition to the $UDPP$, it can be also applied in those situation where it is not currently adopted.
As we have seen in chapter 4, if one wants to develop a slot trading mechanisms for the ATM context two main issues have to be taken into account: the first is to address *equity*; the second, if the mechanism includes an offer-making framework, is the fact that it is very hard to know in advance what is available on the market. In the model we propose these problems are tackled with the following strategy:

- to handle *equity* using AUs preferences as the optimisation target
- rather than establishing a final time schedule, to provide a what-if scenario consisting of a set of possible inter-airline operations that each AU can evaluate

The idea is to reduce, or potentially eliminate, the bias toward flights or airlines characterised by higher total costs, number of passengers, etc., introducing a standardised metric. On the other hand, to allow the possibility of an inter-airline slot trading mechanism, we propose a change of perspective: instead of having airlines formulating offers and the algorithm selecting which ones can take place, the *ISTOP* model tries to match airlines preferences in order to provide a set of slot swaps that each AU involved can either accept or refuse. Before analysing these concepts in details, let us give an overview of how the mechanism would work in practice:

- all airlines can assign an initial priority value to some or each of their own flights involved in the CCS
- in order to better maintain equity, all priority values are then standardised by the usage of a preference function
- an optimisation algorithm produces a set of offers of inter-airline slot exchanges, aiming to better accommodate airline preferences, where an offer consists in a slot exchange between couples of airlines concerning a limited number of their flights
- if the offer is accepted by both airlines the exchange takes place, otherwise they keep the same slots, previously assigned by FPFS or UDPP

In order to provide meaningful results, we require also to operate under the following assumptions:

Assumption 2 *Airlines are interested in accepting an increase of delay of some of their flights which they consider less important, if they can obtain a delay reduction of some of their most important flights in exchange*

Assumption 3 *Priority values are well defined and they can be considered as a good approximation of the actual airline's preferences*

As aforementioned, in the *ISTOP* model, *equity* is mainly managed by designing a suitable objective function that we will now discuss in details.

5.2 Preference and objective functions

In the first two examples of chapter 4, we have seen how an objective function describing the total delay impact can produce a solution that systematically favours certain airlines and disadvantage others. This happens in general if

the objective function weights represent absolute quantities, such as fuel costs, passenger costs, number of passengers etc. The key point is to use relative values, representing in our case, the airlines preferences. Toward this aim we have first to determine preferences in a equitable manner and we require a few definitions:

Definition 2 *$V \in \mathbb{R}^+$ denotes the **priority value** of a flight: priority values are assigned by the airlines in an increasing order, depending on the importance of a flight.*

It is important to notice that no scale has been mentioned. In fact, our goal is to allow airlines to assign priority values using any possible scale. To achieve so, we will introduce two functions, $\mathcal{P}$ and f: the first will aim to standardise the priority values and the second will balance the standardisation, taking into account also the number of flights of the airline that owns the flight for which we are computing the preference.

Definition 3 *Given an increasing monotonic function $f : \mathbb{R}^+ \times \mathbb{N} \to \mathbb{R}^+$, any map $\mathcal{P} : \mathbb{R}^+ \times \mathbb{R}^+ \times \mathbb{N} \to \mathbb{R}^+$ of the form*

$$\mathcal{P}(V, \mathcal{V}, N) = \frac{f(V, N)}{\mathcal{V}} \tag{5.1}$$

*defines a **preference function**.*

Definition 4 *Given an airline A_k with N flights involved in a CCS, a set of positive real numbers $\{V_0, ...V_{N-1}\}$, representing the priority values assigned to the N flights, and defined $\mathcal{V} = \sum_{j \in A_k} V_j$, for each flight $i \in A_k$ the value*

$$p_i = \mathcal{P}(V_i, N, \mathcal{V}) = \frac{f(V, N)}{\mathcal{V}} = \frac{f(V, N)}{\sum_{j \in A_k} V_j} \tag{5.2}$$

*will be called the **preference value** of i*

The function f is essential to determine how *equity* is managed, and many strategies can be explored. In the next example we will show the ones we have chosen to run our simulations.

Example 14 *Given a constant $\alpha \geq 0$, a simple and natural way to define the preference function is:*

$$f(V, N) = V \cdot N^{\alpha}$$

which produces a preference value of this form:

$$\mathcal{P}(V, \mathcal{V}, N) = \frac{V \cdot N^{\alpha}}{\sum_{j \in A_k} V_j}$$

For $\alpha = 0$, p_i will assume values between 0 and 1, meaning that we are operating a full normalisation of the preference values. This choice of α has the

following interpretation. Given an airline A_k, all its flights preference values are normalised as they sum up to one:

$$\frac{\sum_{j \in A_k} V_j \cdot N^0}{\sum_{j \in A_k} V_j} = \frac{\sum_{j \in A_k} V_j}{\sum_{j \in A_k} V_j} = 1$$

If we now think of preference values as credits the role of the function f is equivalent to assigning an equal number of credits to each airline, which can decide how to distribute them among their flights.
For $\alpha = 1$ instead, for each airline the sum of its preference values is equal to N.

$$\frac{\sum_{j \in A_k} V_j \cdot N^1}{\sum_{j \in A_k} V_j} = N$$

Therefore, the resulting role of f, is similar to the previous one, but this time, the number of credits assigned to each airline is proportional to the number of its flights.

Now that we defined all the concepts required, we can better specify how the preference mechanism works:

- to each flight i, the owner airline assigns a priority value V_i
- V_i is then standardised based on a function depending on the value V_i and the number of flights within the CCS held by the same airline. The final result is the actual preference value p_i

Before discussing in details the objective function, let us introduce another little bit of notation required:

- S is set of all slots defining the CCS
- $\mathbb{A}$, the set of all airlines
- A_k will represent a specific airline but we will also refer to it using just the index k, so $A_k \in \mathbb{A}$ will be equivalent to $k \in \mathbb{A}$

There is clearly a correspondence between a slot i, the flight originally assigned to i and the airline $A \in \mathbb{A}$ that owns the flight and holds the slot. Therefore, with a little notation abuse, we will say $i \in A$ referring either to the slot i held by A or the flight i owned by A.
This dualism might appear confusing but it will turn out quite handy when we will need to define the constraints. Lets also introduce:

- $A(i)$, the function that returns the airline which flight i belongs to
- S, the set of all slots
- $c_i \in \mathbb{R}^+$, the cost per minute associated to the $i-th$ flight

- $d_{ij} \in \mathbb{R}^+$, the delay of the flight originally assigned to slot i if assigned to slot j
- $\mathcal{C} : \mathbb{R}^+ \to \mathbb{R}^+$, the function that produces the value $\mathcal{C}(c_i, d_{ij})$, which represents the cost of assigning to flight i a delay d_{ij}
- $ETA(i)$, the function returning the slot index representing the expected arrival time of flight i
- V_i , the priority value assigned to the flight i
- $p_i = \mathcal{P}(V_i, N_k)$, preference value of the flight i, where N_k represents the number of flights within the CCS, owned by the airline A_k

and finally, the decision variables:

- $x_{ij} \in 0, 1$, which means that the flight originally assigned to slot i is assigned to slot j

We can now introduce the objective function. In the next section we will see how formalise the constraints to guarantee the assignment of flights to slots, in order to define inter-airlines offers. But first, we need to ensure that our optimisation algorithm works in accordance to the preferences. In particular we would like to penalise the assignment of significant delays to flights with high preferences. In this respect, as preferences are, up to a certain extent, a relative representation of fights costs, we can provide to the model the possibility to include non linear relations between the delay and the preference penalty. Toward this purpose we can give the following quite general:

Definition 5 *Given p_i, the preference value of a flight i, and a delay d_{ij}, any function $\mathcal{PPF} : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$ with arguments p_i and d_{ij}, will be called* ***preference penalty function***

This allows us to introduce the following:

Definition 6

$$\mathcal{PS} := \sum_{i,j \in S} \mathcal{PPF}(p_i, d_{ij}) \cdot x_{i,j} \tag{5.3}$$

determines the ***(preference) score function****,*

which consequently provides the objective function of the model:

Definition 7

$$\boldsymbol{OBJ} := \min \mathcal{PS} \tag{5.4}$$

So far, we developed our strategy to manage *equity*. The second target is to introduce the possibility of inter-airlines operations.

5.3 The slot trading mechanism

One of the strong enhancements furnished by the *UDPP* mechanism is its capability to provide a what-if scenario that any airline can evaluate "in place", before committing to any schedule change. This characteristic results in practice quite successful, as, how we have seen in section 1.5, the real costs and the various delay-costs function estimations represent just approximations of the actual situation. *UDPP* does not rely on any of these quantities to determine a new schedule, instead, it let airlines decide "on the spot" how to better accommodate their needs. One of the significant limitations of this framework is that it does not allow operations of type T, and this leaves room for further improvements. Our goal then, is to extend this possibility, but trying to maintain the same philosophy. In particular, we would like to provide to each AU, a what-if scenario consisting of a set of possible inter-airlines slot swap offers concerning their flights, where each offer can be evaluated by the corresponding AU and, if considered sufficiently convenient, accept it. It is important to notice that in this context, a feasible inter-airline slot swap is the resulting match of several offers, potentially regarding multiple airlines. Therefore, a trade can actually take place, *iff* all the offers which define the trade are accepted. If one of these offers is rejected by some airline, the trade falls through and all flights included keep their original slot (the one assigned by the *UDPP*, or by the *FPFS* if *UDPP* has not been deployed). As a consequence, in order to make this mechanism applicable in a practical scenario, the number of airlines involved in a single trade has to be limited. To better understand the reason of such limitation, the slot trading graph interpretation seen in section 3.3, turns out quite useful.

Example 15 *Let us suppose to have a hotspot with* 15 *flights and* 3 *airlines, A, B and C, where we assume that a set of feasible and convenient offers has been made for some of the flights. Figure 5.1 shows the graph representation of the trade. Let us also imagine that the red arrows in the figure indicate the optimal solution obtained with some algorithm. This solution consists in a single trade, which is the resulting match of the offers depicted on the right hand side of the figure. Now, if airline B rejects offer* 3*, the entire trade falls through and this implies that no change to the original schedule is made and, consequently, no benefit for any of the airlines involved is achieved. In general, the larger the potential number of airlines involved in a single trade, the lower is chance of the trade to take place, as just one offer refusal is sufficient to stop the entire operation. Therefore, to include the possibility of offers refusal and at the same time enhance the chances of successful trades, the limitation of the trade size, is crucial. For this reason, in this work we just consider the case of* ***one-to-one airline slot trading****. The same consideration can be made about the number of flights included in a single offer. In our formulation we extended the possibility to make offers with any number of flights greater than two, but in our simulations we considered just two flights offers, for computational reasons.*
If in the example we now consider just one-to-one airline trades, a possible solution will look like the one shown in figure 5.2.

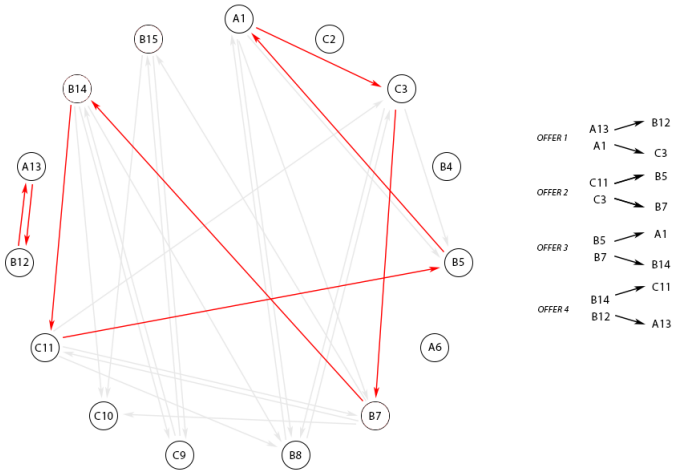


Figure 5.1: Graph representation of possible convenient slot swaps

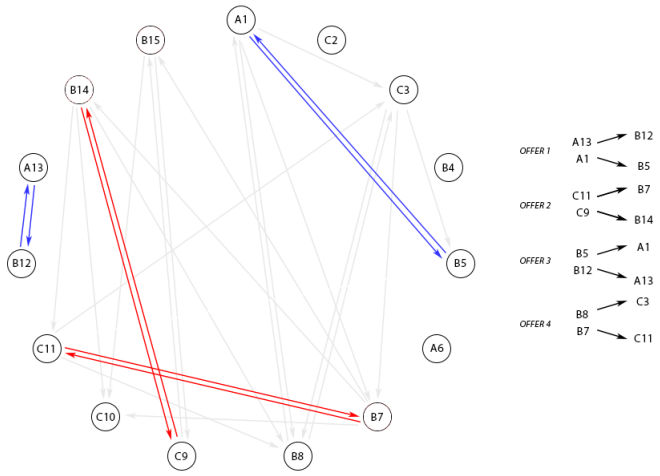


Figure 5.2: Graph representation of possible convenient slot swaps

The resulting two trades are represented with the red and the blue arrows. In this case, if for instance airline C rejects offer 2, the trade between C and B falls through, but this does not effect the trade between A and B, which is still valid.

We can now introduce the constraints defining the model:

Constraint 2 *All flights have to be assigned either to their initial slot or to a slot owned by another airline*

$$\sum_{j \notin A_k} x_{ij} + x_{ii} = 1 \quad \forall i \in A_k,\ \forall k \in \mathbb{A} \tag{5.5}$$

$$\sum_{j \in A_k,\ j \neq i} x_{ij} = 0 \quad \forall i \in A_k \tag{5.6}$$

Constraint 3 *No flight can be assigned to a slot earlier than its expected arrival time*

$$x_{ij} = 0 \quad \forall j \in S :\ j < ETA(i), \quad \forall i \in S \tag{5.7}$$

Constraint 4 *All slots can host at most a single flight*

$$\sum_{i \in S} x_{ij} \leq 1 \quad \forall j \in S \tag{5.8}$$

Now we have to define the concept of offer for this model. In particular we want to interpret an offer as a possibility for two airlines to exchange a certain amount of their slots (couples, triplets, ...,N-tuples). Toward this purpose we can now define:

- T^k, the set of all tuples of size at most N, of slots owned by an airline A_k.

Definition 8 *An **offer** consists of an exchange of tuples of slots of the same size, between two airlines; so if T_s^k belongs to A_k and $T_{s'}^w$ belongs to A_w, the offer of exchanging T_s^k and $T_{s'}^w$ can be represented as:*

$$T_s^k \sim T_{s'}^w$$

Example 16 *Let us suppose that we have 8 flights in a CCS. Let us also imagine that airline A_k owns flights (and so slots) 0,4,7 and airline A_w owns flights 1,2,5.*

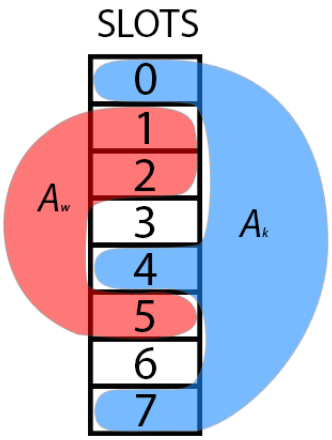


Figure 5.3: Slot configuration

The composition of the sets T^k and T^w is then the one depicted in figure 5.4.

T^k		T^w	
{0,4}	T^k_0	{1,2}	T^w_0
{0,7}	T^k_1	{1,5}	T^w_1
{4,7}	T^k_2	{2,5}	T^w_2
{0,4,7}	T^k_3	{1,2,5}	T^w_3

Figure 5.4: Tuples of airlines A_k and A_w

The potential offers between airlines A_k and A_w are then all the combinations of same length tuples of T^k and T^w, and is shown in figure 5.5

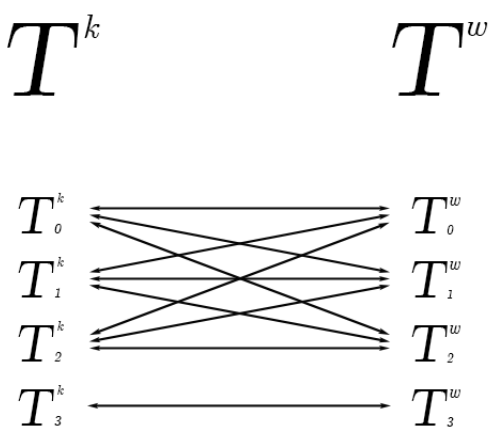


Figure 5.5: Potential offers between airlines A_k and A_w

Finally, we notice that an offer includes multiple swap possibilities; for instance offer $T^k_1 \sim T^w_0$ contains the following slot exchanges:

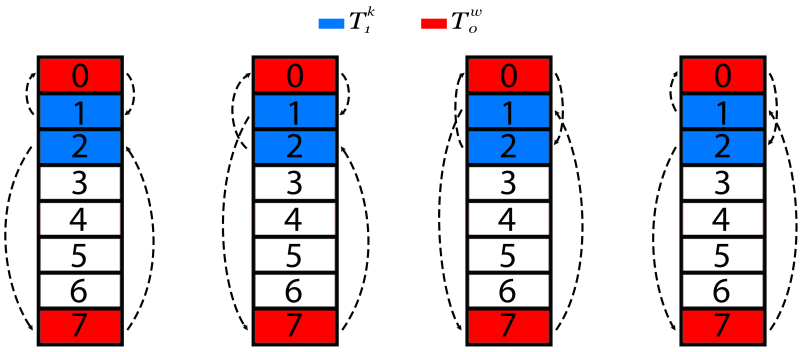


Figure 5.6: Possible swaps for offer $T^k_1 \sim T^w_0$

Let now

- $\mathcal{O}$ be the set of all offers. As long as offer $T^k_s \sim T^w_{s'}$ coincides with offer $T^w_{s'} \sim T^k_s$, both this writings will be representing the same element of $\mathcal{O}$.

Also, the fact that an offer is a match of two tuples of the same size, allows us to define

$$\left| T^k_s \sim T^w_{s'} \right| := |T^k_s| = \left| T^w_{s'} \right| .$$

We can now introduce the decision variables

- $o_{ss'}^{kw} \in \{0,1\}$, which signifies that the offer $T_s^k \sim T_{s'}^w$ has been selected. We will refer to these variables as **offer variables**.

An important **remark**: we have to establish a relation between the decision variables x_{ij} and the offer variables. Let us consider the offer $T_1^k \sim T_0^w$, so $\{0,7\} \sim \{1,2\}$ of example 16, we would like that:

$$\begin{aligned} T_1^k \sim T_0^w \iff \sum_{\substack{i \in T_1^k \\ j \in T_0^w}} x_{ij} = \sum_{\substack{i \in T_0^w \\ j \in T_1^k}} x_{ji} = \mid T_1^k \mid = \mid T_0^w \mid = \mid T_1^k \sim T_0^w \mid = 2 \\ \implies o_{1,0}^{kw} = 1, \end{aligned} \tag{5.9}$$

which means that if offer $o_{1,0}^{kw}$ has been selected then all flights in T_1^k have been assigned to slots in T_0^w and all flights in T_0^w have been assigned to T_1^k. To formulate this condition a few steps are required. First we need to introduce:

- $O(i)$, the function that returns the set of offers where flight i appears

We first want to ensure that if a flight $i \in A_k$ is moved from its original slot, i.e. $x_{ij} = 1$ for some $j \in S$, with $j \neq i$, some offer involving flight i has been selected. We have then to force the corresponding offer variable to be equal to one:

Constraint 5 *If $x_{ij} = 1$, for some $j \neq i$, then it must exist a tuple, including flight i, that has been selected for some offer, which implies that the correspondent offer variable is equal to one:*

$$\sum_{j \in S,\, j \neq i} x_{ij} = \sum_{k \in O(i)} o_k \quad \forall i \in S \tag{5.10}$$

Observation 2 *Due to constraint 2 and 5 we have:*

$$1 \geq \sum_{j \in S,\, j \neq i} x_{ij} = \sum_{k \in O(i)} o_k \quad \forall i \in S, \tag{5.11}$$

which means that we are already ensured that the number of offers selected per flight cannot be greater than one.

The final step to guarantee and generalise the condition wanted in 5.9 is given by:

Constraint 6

$$\sum_{\substack{i \in T_s^k \\ j \in T_{s'}^w}} x_{ij} + \sum_{\substack{i \in T_s^k \\ j \in T_{s'}^w}} x_{ji} \;\geq\; 2 \cdot \mid T_s^k \sim T_{s'}^w \mid \cdot o_{ss'}^{kw} \quad \forall\, T_s^k \sim T_{s'}^w \in \mathcal{O}. \tag{5.12}$$

Observation 3 *Constraints 5 and 6 actually achieve our goal. If* $o^{kw}_{ss'} = 1$ *we have:*

$$\sum_{\substack{i\in T^k_s\\ j\in T^w_{s'}}} x_{ij} + \sum_{\substack{i\in T^k_s\\ j\in T^w_{s'}}} x_{ji} \;\geq\; 2 \cdot \mid T^k_s \sim T^w_{s'} \mid,$$

but

$$\sum_{\substack{i\in T^k_s\\ j\in T^w_{s'}}} x_{ij} \leq \mid T^k_s \mid \quad \textit{and} \quad \sum_{\substack{i\in T^k_s\\ j\in T^w_{s'}}} x_{ji} \leq \mid T^w_{s'} \mid .$$

Also, $\mid T^k_s \mid = \mid T^w_{s'} \mid = \mid T^k_s \sim T^w_{s'} \mid$ *as offers are made of tuples of the same size. This implies:*

$$\sum_{\substack{i\in T^k_s\\ j\in T^w_{s'}}} x_{ij} = \sum_{\substack{i\in T^k_s\\ j\in T^w_{s'}}} x_{ji} \;= \mid T^k_s \sim T^w_{s'} \mid,$$

In addition we have to ensure that if a slot swap occurs, it has to be mutually beneficial for both airlines involved.

Constraint 7

$$\sum_{\substack{i\in T^k_s\\ j\in T^w_{s'}}} x_{ij} \cdot \mathcal{C}(c_i, d_{ij}) - (1 - o^{kw}_{ss'}) \cdot \mathcal{M} \leq \sum_{\substack{i\in T_s\\ j\in T_{s'}}} x_{ij} \cdot \mathcal{C}(c_i, d_{ii}) - \varepsilon \quad \forall\, T^k_s \sim T^w_{s'} \in \mathcal{O}; \tag{5.13}$$

$$\sum_{\substack{i\in T_s\\ j\in T_{s'}}} x_{ji} \cdot \mathcal{C}(c_j, d_{ji}) - (1 - o^{kw}_{ss'}) \cdot \mathcal{M} \leq \sum_{\substack{i\in T^k_s\\ j\in T^w_{s'}}} x_{ji} \cdot \mathcal{C}(c_j, d_{jj}) - \varepsilon \quad \forall\, T^k_s \sim T^w_{s'} \in \mathcal{O} \tag{5.14}$$

where $\mathcal{M} >> 0$ *and* $\varepsilon > 0$ *are appropriate dummy constants.*

To better understand how constraint 7 works, a few considerations are essential.

Observation 4 *Let us consider just equation 5.13. Suppose that the offer* $T^k_s \sim T^w_{s'}$*, between* A_k *and* A_w*, has been selected. Let now* $\mathcal{T}$ *be the set of couples of indexes* (i, j)*, with* $i \in T^k_s$ *and* $j \in T^w_{s'}$*, for which* $x_{ij} = 1$*.* $\mathcal{T}$ *is basically representing the actual trade, as an element* (i, j) *of* $\mathcal{T}$ *states that flight* i *in* T^k_s *has been assigned to slot* j *in* $T^w_{s'}$*. Constraint 6 ensures that* $\mid \mathcal{T} \mid = \mid T^k_s \sim T^w_{s'} \mid$*. At this stage the left hand side of constraint 7 becomes:*

$$\begin{aligned}\sum_{(i,j)\in\mathcal{T}} x_{ij} \cdot \mathcal{C}(c_i, d_{ij}) - (1 - o^{kw}_{ss'}) \cdot \mathcal{M} &= \sum_{(i,j)\in\mathcal{T}} 1 \cdot \mathcal{C}(c_i, d_{ij}) - (1 - 1) \cdot \mathcal{M} \\ &= \sum_{(i,j)\in\mathcal{T}} \mathcal{C}(c_i, d_{ij})\end{aligned}$$

which represents the cost to move the flights of A_k currently assigned to slots in T_s^k, to slots in $T_{s'}^w$ with the particular allocation defined by $\mathcal{T}$. The right hand side instead, becomes:

$$\sum_{(i,j)\in\mathcal{T}} x_{ij}\cdot\mathcal{C}(c_i,d_{ii})-\varepsilon=\sum_{(i,j)\in\mathcal{T}} 1\cdot\mathcal{C}(c_i,d_{ii})-\varepsilon=\sum_{(i,j)\in\mathcal{T}}\mathcal{C}(c_i,d_{ii})-\varepsilon$$

that is the cost of keeping all flights of A_k currently assigned to slots in T_s^k, in their original slots, minus a small constant; applying the inequality given by constraint 7, we eventually have:

$$\sum_{(i,j)\in\mathcal{T}}\mathcal{C}(c_i,d_{ij})\leq\sum_{(i,j)\in\mathcal{T}}\mathcal{C}(c_i,d_{ii})-\varepsilon\implies\sum_{(i,j)\in\mathcal{T}}\mathcal{C}(c_i,d_{ij})<\sum_{(i,j)\in\mathcal{T}}\mathcal{C}(c_i,d_{ii})$$

which means that if the flights of A_k currently assigned to slots in T_s^k, are moved to slots in $T_{s'}^w$ with the allocation defined by $\mathcal{T}$, this implies a cost reduction for A_k. Following the same reasoning for equation 5.14 we would then obtain:

$$\sum_{(j,i)\in\mathcal{T}'}\mathcal{C}(c_j,d_{ji})<\sum_{(j,i)\in\mathcal{T}'}\mathcal{C}(c_j,d_{jj})$$

for some set $\mathcal{T}'$ of couples of indexes (j,i) ,with $j\in T_{s'}^w$ and $i\in T_s^k$, which states that if the flights of A_w currently assigned to slots in $T_{s'}^w$, are moved to slots in T_s^k, this implies a cost reduction also for A_w. In conclusion we achieved that if offer $T_s^k\sim T_{s'}^w$ is selected, the corresponding slot swaps would result mutually beneficial for both airlines involved.

5.4 Customising the $\mathcal{C}$ function

In our simulations, the function $\mathcal{C}$ is assumed to represent the real relation between delay and costs. This, due to constraint 7, ensures that the offers provided are convenient for the respective AUs. However, we are also implicitly assuming that the same cost function is describing all airlines cost behaviour, which in a practical scenario it is a quite unrealistic assumption. For this issue, there is fortunately a work around, which actually turns out to be a very interesting potential feature to improve the accuracy and the flexibility of the model. If we look at constraint 7, we notice that $\mathcal{C}$ operates just on parameters relative to a single airline and the resulting condition, is again referring to the same airline. In some sense, we can then say that each airline "owns" this constraint. This fact, allows us to extend the possibility to "customise" it, and let each airline define its own cost function. If we then introduce:

- $\mathcal{C}_k$, the cost function relative to airline $k\in\mathbb{A}$

equation 5.13 (or equivalently 5.14) of constraint 7, can be rewritten as follow:

Constraint 8

$$\sum_{\substack{i\in T_s^k \\ j\in T_{s'}^w}} x_{ij}\cdot \mathcal{C}_k(c_i,d_{ij}) - (1-o_{ss'}^{kw})\cdot\mathcal{M} \leq \sum_{\substack{i\in T_s \\ j\in T_{s'}}} x_{ij}\cdot \mathcal{C}_k(c_i,d_{ii}) - \varepsilon \qquad (5.15)$$

$$\forall\, T_s^k \sim T_{s'}^w \in \mathcal{O};$$

where $\mathcal{M} >> 0$ *and* $\varepsilon > 0$ *are appropriate dummy constants.*

Furthermore, airlines are generally reluctant to declare their real costs and even in this last formulation, the parameters c_i are still required. This issue can be fixed if we decide to work directly with preference values rather then costs. In this case, instead of the cost functions $\mathcal{C}_k$, all airlines will have to provide a *comparison function*, $\mathcal{CO}_k$. The role of $\mathcal{CO}_k$ is identical to one of $\mathcal{C}_k$, which is to establish whether a potential offer is convenient or not, but this time, based on delays and preferences. The new formulation of constraint will then be:

$$\sum_{\substack{i\in T_s^k \\ j\in T_{s'}^w}} x_{ij}\cdot \mathcal{CO}_k(c_i,d_{ij}) - (1-o_{ss'}^{kw})\cdot\mathcal{M} \leq \sum_{\substack{i\in T_s \\ j\in T_{s'}}} x_{ij}\cdot \mathcal{CO}_k(c_i,d_{ii}) - \varepsilon \qquad (5.16)$$

$$\forall\, T_s^k \sim T_{s'}^w \in \mathcal{O};$$

Toward a further development of the *ISTOP* for a real scenario application, the *comparison function* represents a crucial feature: as we already mentioned costs are normally hard to estimate, and even in those cases in which they are known, airlines prefer to keep them confidential as they are considered sensitive information regarding their business strategy. This is one of the reasons of the success of *UDPP*, which does not require any knowledge of costs as it operates just with preferences. This fact still introduces some approximations as preferences represent, up to a certain extent, the economical value of the flights. But here, the *what if scenario* framework provided by the *UDPP*, furnishes a good compromising solution between the airlines need to improve their situation and the respect of the confidentiality of costs. In this perspective, the *comparison function* allows the *ISTOP* model to perfectly embrace the *UDPP* philosophy. The *ISTOP* can be then interpreted as an airline's "broker" that, given the preferences and the *comparison function*, checks what is available on the market and provides a set of possible offers on which the airline will have the "last word". Our simulations were costs based as the target was simply to explore the potential of the model, reducing as much as possible the error sources: using costs to determine preferences allowed us to ensure an optimal preference assignment and therefore to better evaluate the model performance. A more advanced study on the preference assignment dynamics is certainly a subject for future research, but at this stage it remains out of the scope of this work.

Chapter 6

Simulations

The targets of our simulations were:

- test the potential of the UDPP mechanism, assuming that all AUs participate in UDPP and that they all adopt the preference assignment provided by the model discussed in 4.5.3
- test the capability of our model to improve the UDPP solution in terms of total delay impact reduction, under the assumption that all the offers found by the algorithm were automatically accepted by the AUs

In order to evaluate the results, the *max reduction* model has been tested. Its solution has been considered as an upper bound of the total cost reduction and so as term of comparison for the other models. The simulations have been then performed with the following procedure:

- we randomly generated an initial schedule, which was assumed to be the FPFS solution. From this initial schedule we computed the Max reduction solution to provide the upper bound solution
- we computed the UDPP solution starting from the initial schedule
- we computed our model solution starting from the UDPP solution.

Different configurations in terms of number of flights, number of airlines and number of flights per airline have been considered. For each configuration the above procedure has been repeated 50 times. Starting from the data collected, we finally produced a statistical analysis. The implementation has been made in Python using the linear optimisation library MIP.

6.1 The test scenarios

For our simulations we decided to imagine two scenarios:

- a hotspot concerning 50 flights and involving 15 airlines with a 50% capacity reduction
- a hotspot concerning 70 flights and involving 20 airlines with a 50% capacity reduction

To analyse the effects of the various algorithms on the AUs, the number of flights per airline has been chosen using the empirical distribution in order to cover a good variety of cases and guarantee in each configuration a consistent number of LVOCs. The average configurations for the two cases are shown in figure 6.1. The initial ordering of flights has been randomly assigned.

6.2 Parameter settings

For all models, the cost function we used was the one introduced in section 1.5:

$$\mathcal{C}(c_f, d_f) = \frac{c_f \cdot d_f^2}{2} \tag{6.1}$$

where c_f is the cost per minute of delay of flight f and d_f its minutes of delay. We remark again that this quadratic model represents a strong simplification, but for the purposes of our tests it could be considered as a sufficient approximation, as our main concern was to capture the non linear relation between delays and costs. To assign to a flight i its parameter c_i we followed a similar strategy to the one adopted by Ruiz et al. Ruiz et al. [2019], choosing a value in the interval $]0.5, 2[$ with this procedure:

- sample from $\{0, 1\}$ with equal probability, then:
  - if 0, sample a value from a truncated normal distribution with $\mu = 0.7$ and $\sigma = 0.1$
  - if 1, sample a value from a truncated normal distribution with $\mu = 1.5$ and $\sigma = 0.1$

This mechanism allowed us to divide the flights in two categories, *high* costs and *low* costs, but still including some variability.

To test our model, we decided to set the *priority values* equal to the respective flights cost per minute, i.e. for any flight i, then $V_i = c_i$. This choice has been made to ensure a sensible priority assignment, under the assumption that costs are generally representative of the actual AUs preferences.
For the function f we decided to use the one introduced in example 14, so:

$$f(V, N) = V \cdot N^\alpha$$

which provided the resulting *preference function*:

$$\mathcal{P}(V, \mathcal{V}, N) = \frac{V \cdot N^\alpha}{\sum_{j \in A_k} V_j}$$

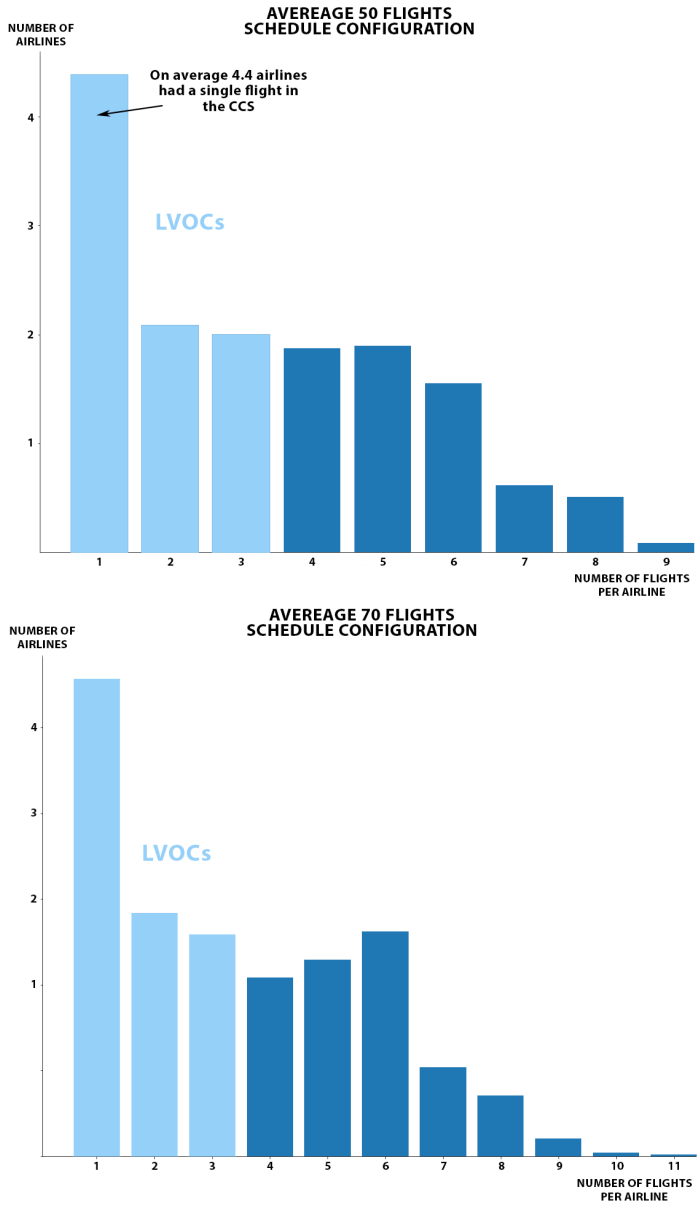


Figure 6.1: Average schedule configurations

testing the values of α in $\{0, 0.5, 1\}$.
As long as we have defined the *priority values* equal to the costs, we decided to use the cost function also as *preference penalty function*, which means:

$$\mathcal{PPF} = \mathcal{C}.$$

6.3 Results

To analyse the results we obtained from the simulations, we considered three main aspects:

- total delay impact reduction
- distribution of the offers among the airlines
- distribution of the impact reduction among the airlines

It is important to remark that we are interested in the **average** benefits, either in terms of costs reduction or number of offers, provided by the three models regardless of how they are distributed among the different simulations. In other words, if we are for example interested in the costs reduction, we want to see what are the long term effects without considering the variability within the episodes; for this reason the **variance** will not be a parameter taken into account for our evaluations.

6.3.1 Total impact reduction

Figure 6.2 shows in summary the total costs reduction of the three algorithms. We can notice that in each case the *ISTOP* model provides an improvement with respect to the UDPP solution. It can be argued that the difference remains quite limited, especially if we compare the total impact reduction performed by the *max reduction* algorithm; however we have to remember that the *max reduction* operates under very weak equity constraints. In this sense, the main result is that we proved that within a CDM framework that respects the UDPP "philosophy" some improvements of the UDPP solution are still possible.

6.3.2 Distribution of the offers among the airlines

The total impact reduction is just a partial metric as to evaluate the performances from the equity perspective; it is also important to consider how the reduction is distributed among the airlines. In particular, as long as the *ISTOP* produces offers, if we gather together airlines by number of flights, we might be interested in seeing how the offers are distributed among the different classes. In the 50 flights tests we had 9 classes of airlines, from the first class which included airlines with just one flight to class 9, which contained the airlines with 9 flights. Figure 6.1 shows the average number of airlines per class in a schedule. We remark that the largest class is the one including airlines with

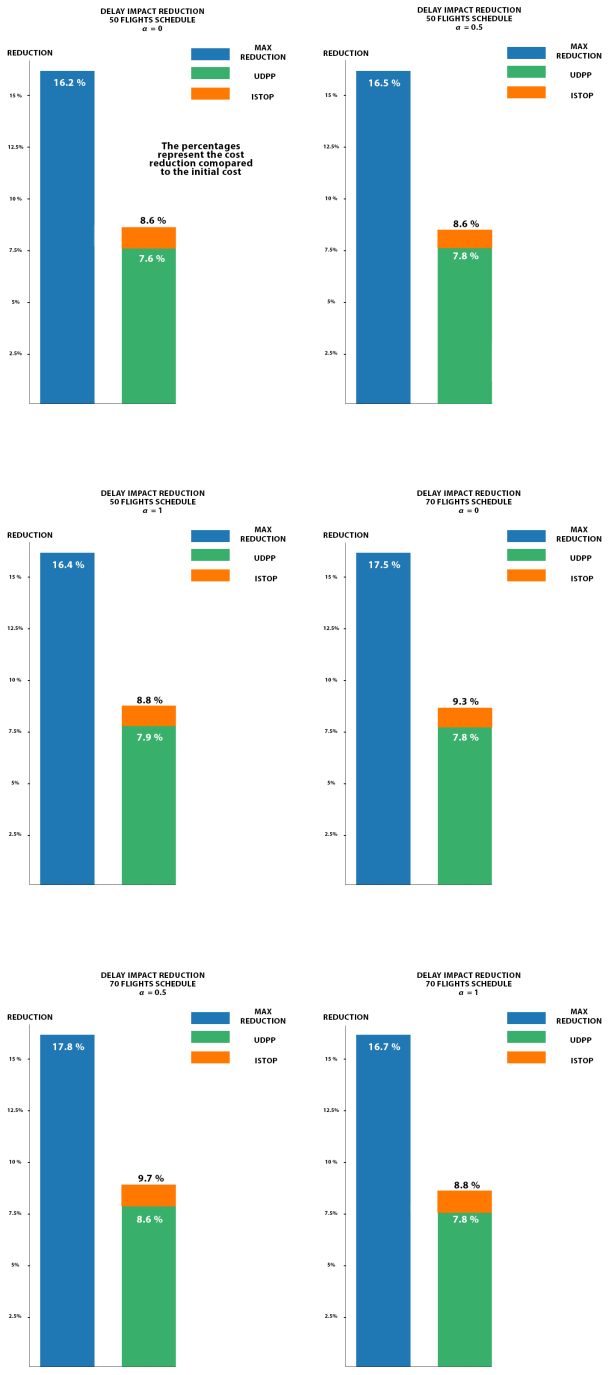


Figure 6.2: Total impact reduction

just one flight: this represents a strong limitation for both UDPP and *ISTOP* efficiency as both cannot provide any benefit to AUs with just a single flight. In particular, this also restricts the chances for our model to find feasible offers for other airlines. However, the presence of multiple airlines with a single flight in a hotspot is a common situation in a real scenario and it must be considered. As shown in figures 6.3 and 6.4, the various choices of α do not produce any significant difference as in all cases the offer distribution is characterised by a bell shape, which means that in general the middle classes receive on average more offers. However, it is important to remark that classes 2 and 3, which can be defined as LVOCs in both the 50 and 70 flights schedules, generally share around the 10% of the offers which still represents a not negligible improvement.

In particular, if on one hand the number of offers relative to classes 2 and 3 remains limited, on the other, their effects in terms of cost reduction represent a more significant improvement especially if we consider that UDPP generally provides modest benefits to LVOCs.

6.3.3 Considerations

One of the target of the simulations was to test the efficiency of the UDPP under the assumption that all airlines are computing their preferences in an optimal manner. The result can be then interpreted as an upper bound to the performance of the UDPP. As figure 6.2 shows, the *max reduction* model is significantly stronger in terms of total impact reduction. However, it is hard to quantify how much the equity constraints represent a limitation to the total delay impact reduction. Despite the tests of the *ISTOP* model simply aimed to perform a first exploration of its potential, we certainly proved that it is possible to provide some improvement to the UDPP framework, also respecting its fairness principles. Nevertheless, deeper studies on the parameter settings are still required and different choices of cost functions and preference functions can be tested.

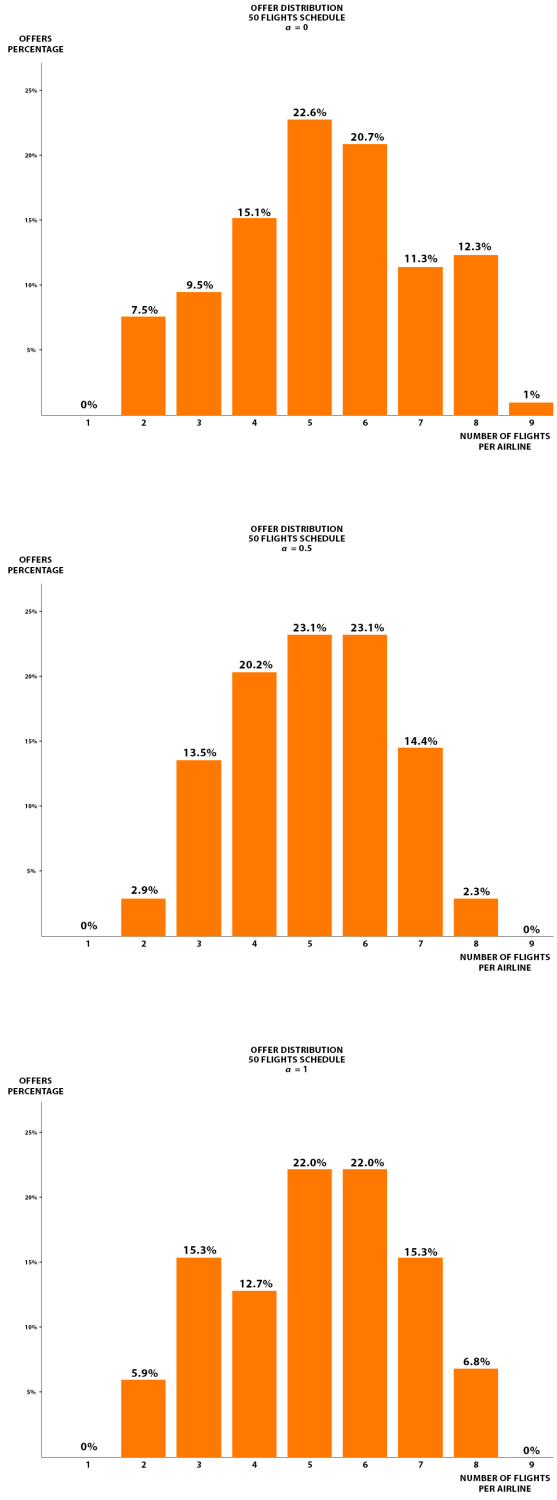


Figure 6.3: Distribution of the offers in the case of the 50 flights schedule

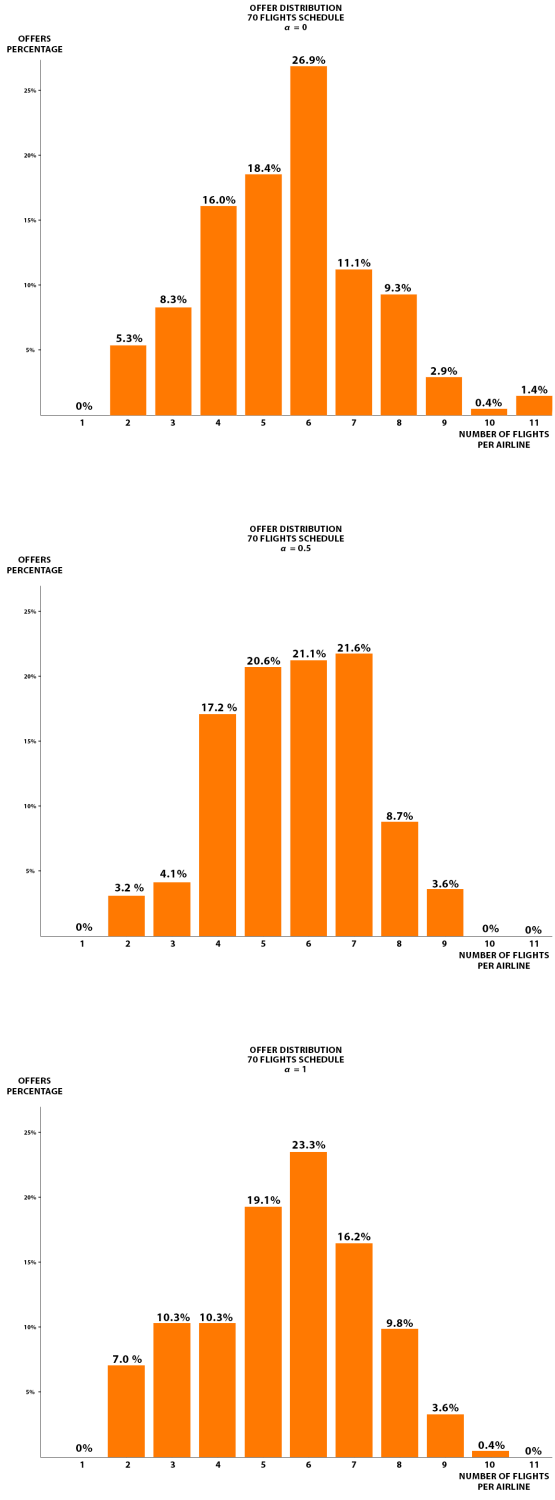


Figure 6.4: Distribution of the offers in the case of the 70 flights schedule

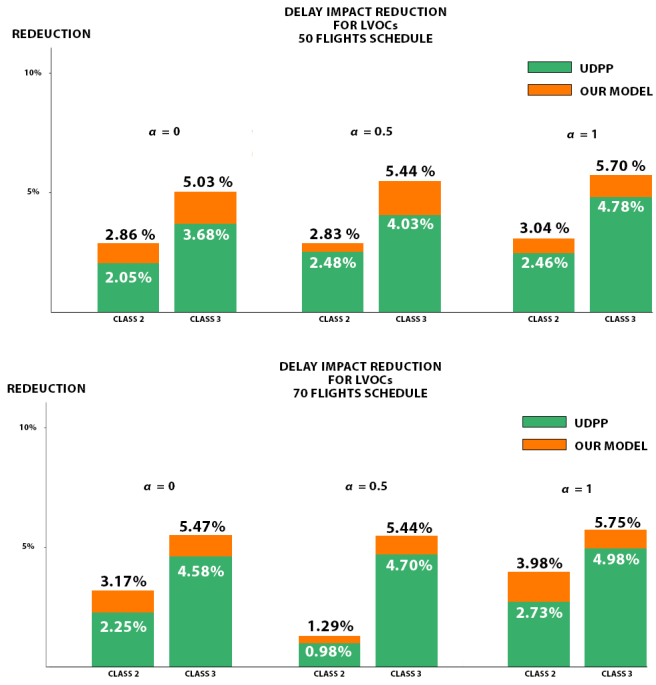


Figure 6.5: Delay impact reduction for LVOCs

Chapter 7

Conclusions

In this work we analysed the delay impact problem caused by a CSS affecting an airport, with a particular focus on the *equity* concerns. We have seen also how, in this respect, the CDM represents a keyword to overcome *fairness* issues, and how UDPP perfectly captures and exploits this principle. However, we showed that it is still possible to improve this mechanism. Currently, the preferences assignment is empirically managed by the AUs, and especially with a great number of flights, an efficient preference setting might be difficult to obtain. The fact that any UDPP local solution can be equivalently obtained by an appropriate usage of the only FDR and SFP features, allowed us to develop a model to mathematically optimise this local solution. On the other hand, the model we proposed in section 4.5.3 is based on the assumption that costs and the cost function are known, and, as we have seen in section 1.5, these estimations are in most cases just approximations of the real scenario. Therefore, entirely rely only on them for the preference assignment represents a strategy that some AU might consider too risky. But here, the UDPP structure and its *what if scenario* framework, can be exploited: the model can be simply used to obtain an initial preference setting, and the correspondent local solution can be interpreted as a baseline solution. Were the model did not capture the actual AUs expectations, preferences can be dynamically adjusted and a further local solution can be obtained, and once again evaluated. Basically, the more accurate are the costs and the cost function estimations, the quicker this procedure would lead to a satisfying solution. Moreover, the algorithm can be used directly by the airlines, and this allows its usage also to those AUs that want to keep their costs secrets.

Another aspect that we analysed in detail is the fact that UDPP does not allow operations of type $\mathcal{T}$; this signifies that for some AUs, there is potentially still room for further delay impact reduction. In particular this is the case of the LVOCs. Toward this aim we proposed a model to extends the possibility of inter-airline slot trading within the UDPP context. To embrace the UDPP philosophy, we have chosen to base the optimisation on preferences, which makes

the equity management more flexible. The model includes also a specific constraint to guarantee no negative impact to any AU. To ease the computation, in our simulations we used costs to ensure this condition but we also provided an additional formulation that allows to work with preference only. This makes this framework suitable also for those airlines which are reluctant to declare their costs. In order to operate in accordance with the CDM paradigm, the model we developed, instead of a final schedule, aims to provide a what if scenario, consisting of a set of offers, that each AU can decide whether to accept or refuse. This last characteristics is crucial to make this mechanism applicable in a real situation. As we said, costs estimations, and consequently preferences, are just approximations of the real delay impact and what the model does is basically to exploit this approximation and the mathematical optimisation to explore which improvements of the UDPP solution are still achievable. The last word is then given back to the AUs, which can evaluate the offers received and make the final decision. In addition, this particular framework provides a potential additional tool to reduce delay impact also for LVOCs, which are normally not in the condition to exploit UDPP due to their limited number of flights.

The results obtained in our simulations leave room for several considerations. First of all we showed that the UDPP mechanism can potentially guarantee good performances if an optimal preference assignment policy is followed. With this respect, the model we developed for the UDPP local solution optimisation, results crucial as it provides the optimal preference assignment. In addition, applying our slot trading model starting from the UDPP solution as initial condition, we proved that this performance can be further improved if one allows inter-airline slot trading dynamics, and that our solution can also provide more opportunities for the LVOCs to reduce their costs. The results we achieved with our simulations are quite promising but further tests are needed: we have seen how the average number of inter-airlines operations found by the *ISTOP* model in order to improve the UDPP solutions tends to grow with the number of flights in the CCS, but also that in certain situations it remained quite limited. One of the possible reasons of this limitation might be related to the particular parameters chosen and the assumptions on the flights costs and the cost function. Different choices might lead to better results and this is certainly a subject for a future research.

We finally remark that this works represents just an initial exploration of a framework that we consider promising. A detailed study on how to choose the trading model objective function and what are the effects in terms of equity management is certainly needed. A more advanced research on how to efficiently define preferences is also required, in particular if one wants to further develop the custom constraint framework introduced in 5.4, where the comparison function also represents an important aspect to investigate.

Bibliography

Amedeo R Odoni. The flow management problem in air traffic control. In *Flow control of congested networks*, pages 269–288. Springer, 1987.

Mostafa Terrab and Amedeo R Odoni. Strategic flow management for air traffic control. *Operations research*, 41(1):138–152, 1993.

Octavio Richetta and Amedeo R Odoni. Solving optimally the static ground-holding policy problem in air traffic control. *Transportation science*, 27(3): 228–238, 1993.

Dimitris Bertsimas and Sarah Stock Patterson. The air traffic flow management problem with enroute capacities. *Operations research*, 46(3):406–422, 1998.

Dimitris Bertsimas, Guglielmo Lulli, and Amedeo Odoni. An integer optimization approach to large-scale air traffic flow management. *Operations research*, 59(1):211–227, 2011.

Thomas Vossen and Michael Ball. Optimization and mediated bartering models for ground delay programs. *Naval research logistics (NRL)*, 53(1):75–90, 2006a.

Thomas WM Vossen and Michael O Ball. Slot trading opportunities in collaborative ground delay programs. *Transportation Science*, 40(1):29–43, 2006b.

Hanif D Sherali, Justin M Hill, Michael V McCrea, and Antonio A Trani. Integrating slot exchange, safety, capacity, and equity mechanisms within an airspace flow program. *Transportation Science*, 45(2):271–284, 2011.

Dimitris Bertsimas and Shubham Gupta. Fairness and collaboration in network air traffic flow management: an optimization approach. *Transportation Science*, 50(1):57–76, 2016.

Nadine Pilon, AJ Cook, Sergio Ruiz, Andrada Bujor, and Lorenzo Castelli. Improved flexibility and equity for airspace users during demand-capacity imbalance-an introduction to the user-driven prioritisation process. *Sixth SESAR Innovation Days*, 2016.

Sergio Ruiz, Laurent Guichard, Nadine Pilon, and Kris Delcourte. A new air traffic flow management user-driven prioritisation process for low volume operator in constraint: Simulations and results. *Journal of Advanced Transportation*, 2019, 2019.

Eurostat, 2020.

Eurocontrol ATM Lexicon, 2020.

Andrew J Cook and Graham Tanner. European airline delay cost reference values, final report (version 3.2). 2011.

Andrew J Cook and Graham Tanner. European airline delay cost reference values, updated and extended values (version 4.1). 2015.

Lorenzo Castelli, Raffaele Pesenti, and Andrea Ranieri. The design of a market mechanism to allocate air traffic flow management slots. *Transportation research part C: Emerging technologies*, 19(5):931–943, 2011.

SESAR. Sesar solution pj07.02 spr-interop/osed for v2 - part. 2019.