UC Irvine

UC Irvine Electronic Theses and Dissertations

Title

Dynamic Network Redesign for Strategic Air Traffic Planning

Permalink

https://escholarship.org/uc/item/2n87q7j0

Author

FRANCI RODON, ARNAU

Publication Date

2017

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA,
IRVINE

Dynamic Network Redesign for Strategic Air Traffic Planning

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Mechanical and Aerospace Engineering

by

Arnau Francí Rodon

Thesis Committee:
Professor Kenneth D. Mease, Chair
Professor Tryphon Georgiou
Professor Haithem Taha

2017

© 2017 Arnau Francí Rodon

TABLE OF CONTENTS

Page

LIST OF FIGURES **v**

LIST OF ALGORITHMS **vi**

LIST OF SYMBOLS AND ABBREVIATIONS **vii**

ACKNOWLEDGMENTS **viii**

ABSTRACT OF THE THESIS **ix**

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Aim of the Project . . . 3
- 1.3 Definition of the Problem . . . 4
  - 1.3.1 Aggregate route model with Eulerian flow for strategic planning . . . 5
  - 1.3.2 Adding flexibility to the ARM for weather contingencies. Overview of the solution proposed . . . 7
  - 1.3.3 Mathematical formulation of the framework . . . 9
- 1.4 Outline of the Thesis . . . 10

2 Avoidance Zones from Weather Forecasts **12**
- 2.1 On Evaluating the Impact of Convective Weather in Air Traffic . . . 13
- 2.2 Generating the Core of the Avoidance Polygons . . . 14
- 2.3 Avoidance Polygons Processing: Clustering the Convex Hulls . . . 16
  - 2.3.1 Independent polygons detection . . . 18
  - 2.3.2 Convex hull calculation . . . 20
- 2.4 From Avoidance Polygons to the No-Fly Zones . . . 20

3 Graph Extension: Generating Feasible Alternative Routes **22**
- 3.1 Constraints for the Graph Extension . . . 24
  - 3.1.1 Kinodynamic constraints . . . 24
  - 3.1.2 Other constraints imposed . . . 25
- 3.2 Recursive Procedure for the Graph Extension . . . 26
  - 3.2.1 Termination condition: determining goal nodes . . . 29
  - 3.2.2 Iterative approach for selecting the start node . . . 29

3.3 Spatio-Temporal Graphs for Time-Varying Edge Costs 31

4 Extended Graph Search: Alternative Routes Selection 35
4.1 Graph Search Algorithm: Lifelong Planning A^* 35
4.1.1 Multiplicity of goals 35
4.1.2 Heuristics 36
4.1.3 Incremental search 37
4.2 Completeness and Correctness of the Algorithm 37

5 Route Network Model as a Linear Time-Invariant System 39
5.1 Air Traffic Flow Uncontrolled Dynamics for a Network Component 40
5.1.1 State Propagation for a network component 40
5.1.2 Scheduled Departures 41
5.1.3 Time-comprehensive state equation for an uncontrolled network component 42
5.2 Air Traffic Flow Controls for a Network Component 43
5.2.1 Ground holding 43
5.2.2 Pre-departure rerouting 44
5.2.3 Airborne rerouting 44
5.2.4 Aggregated controls matrix and input 45
5.3 Whole Network Controlled Dynamics 46

6 The Strategic Planning Problem 47
6.1 Constraints 48
6.1.1 Capacity constraints 48
6.1.2 Model consistency constraints 52
6.2 Cost Function 56
6.3 ILP Problem 58

7 Application of the Model: Fully Worked Basic Example 59
7.1 Description 59
7.1.1 Network description 59
7.1.2 Constraints description 61
7.1.3 Initial layout 62
7.2 Dynamics 62
7.3 Controls 67
7.3.1 Controls input to optimize 67
7.3.2 Controls matrix 68
7.3.3 Network controlled dynamics 69
7.4 Constraints 70
7.4.1 Capacity constraints 70
7.4.2 Model consistency constraints 77

8 Conclusions 80

Bibliography 82

9 Appendix B: Sample Codes in Matlab® 84
9.1 Chapter 2 Algorithms 84
9.1.1 Main Avoidance Zone Generation 84
9.1.2 Closed Polygon Identifier 87
9.1.3 Isolated Convex Hull Calculator 90
9.2 Chapter 4 Algorithms 91
9.2.1 Lifelong Planning A* 91

LIST OF FIGURES

		Page
1.1	ARM generation	6
1.2	Weather blockage for an ARM 1	8
1.3	Weather blockage for an ARM 2	8
1.4	ATM Strategic Planning Tool flowchart.	11
2.1	Unpolished vs. polished avoidance polygons	14
2.2	Avoidance polygons core generation from the weather product.	15
2.3	Avoidance polygons polishing process	17
3.1	Graph extension generation example	28
3.2	Multiple stages of the graph generation	29
3.3	Different graph layouts by tuning the dynamics parameters.	30
3.4	Recursive approach for graph extension to find an alternative route	32
3.5	Recursive approach for graph extension to find an alternative route, continuation	33
3.6	Spatial-temporal graph representation	34
7.1	Network routes and nodes	60
7.2	Air Sectors of the network.	61

LIST OF ALGORITHMS

		Page
1	Main Avoidance Zone Generation	18
2	Closed Polygon Identifier	19
3	Isolated Convex Hull Calculator	20

LIST OF SYMBOLS AND ABBREVIATIONS

ARM	Aggregate Route Model
ATC	Air Traffic Control
ATM	Air Traffic Management
FAA	Federal Aviation Administration
ILP	Integer Linear Programming
LTI	Linear Time Invariant
NAS	National Airspace
O-D	Origin-Destination
$\mathcal{A}$	State matrix
$\mathcal{B}$	Controls matrix
$\mathcal{D}$	Departures matrix
$\mathcal{M}$	Picker matrix
N_{OD}	Number of new routes introduced by the network redesign
N_{OD}	Number of Origin-Destination pairs
N_n	Number of nodes
N_{dep}	Number of departure nodes (first node of a route)
gh	Ground holding control
pr	Pre-departure rerouting control
ar	Airborne rerouting control
$\mathbf{d}$	Departures vector
$\mathbf{u}$	Controls vector
$\mathbf{x}$	State vector, composed by every node of the network
${}^{i}x$	Node i of the network
η	Capacity limit
τ	Weather product threshold for the avoidance zones
$\Omega(\cdot)$	Orthodromic (great-circle) distance function
$\lfloor \cdot \rfloor$	Floor function

ACKNOWLEDGMENTS

First and foremost, my sincere gratitude to Pete Balsells, the University of California, and the Generalitat de Catalunya. They made this invaluable experience possible through the opportunity bestowed.

My heartfelt appreciation to my research group peers, Dr. Alessandro Bombelli and Dr. Eric Trumbauer. This project would have been very different without them. Besides granting me the base knowledge necessary for the project, they have always been ready to help, and most importantly they have become good partners to exchange and discuss ideas; from this fruitful debates has emerged the work here presented. Previous work by Dr. Bombelli on the model has also greatly simplified my endeavor.

I would like to thank my advisor Prof. Kenneth D. Mease, for his willingness to support me on every aspect of the academic life when it was needed, as well as for his supervising role throughout the forging of this work to ensure the global consistency a research project requires.

I want to thank Mr. Adrià Segarra and Mr. Lluís Soler too for introducing me to the topic of Air Traffic Control and the related research ongoing at UCI.

Thanks to everybody who loves me, and thanks to everyone I love.

This material is based upon work supported by the National Aeronautics and Space Administration under Prime Contract Number NAS2-03144.

ABSTRACT OF THE THESIS

Dynamic Network Redesign for Strategic Air Traffic Planning

By

Arnau Francí Rodon

Master of Science in Mechanical and Aerospace Engineering

University of California, Irvine, 2017

Professor Kenneth D. Mease, Chair

This research extends previous work on strategic air traffic planning based on an aggregate route model. Assuming a base model has been developed from historic flight data for clear weather days, what is needed for planning on a particular day with predicted convective weather and possibly other planning horizon dependent capacity constraints, is a means of adding appropriate routes to the model to accommodate the conditions of the planning day.

The route redesign to avoid convective weather is formulated as a path planning problem under differential constraints, which generates feasible trajectories for the aircraft. Special emphasis is placed on the trade-off between obtaining realistic trajectories and maintaining the network low dimensionality characteristic of the aggregate route models. Following previous work, the network dynamics are modeled as a discrete linear time-invariant system, and the minimization of an objective function based on the total delay accumulation is then posed as an integer linear programming problem, with the control inputs as the variable. A test case demonstrates an enhancement of the model's performance when managing a convective weather scenario by a 47% reduction of the total time-delay accumulated respect to the results without the network redesign feature.

Chapter 1

Introduction

1.1 Motivation

To give a taste of the aviation field relevance for the United States, the Federal Aviation Administration (FAA) estimated the total economic impact from civil aviation in 2008 at 1.3 trillion dollars, roughly 9% of the country's Gross Domestic Product for the same year [2]. Flight delays place a significant strain on the US air travel system and cost airlines, passengers, and society billions of dollars each year. Accounting just for direct costs, the 2008 cost was 31 billion dollars.

Besides purely operational costs, air traffic impacts many other areas. In 2016, the global aviation industry produced around 2% of the total CO2 emissions [25], and delays materialized in terms of inefficient reroutes or holding patterns increase the amount of fuel consumed. If we consider the total economic impact of delays through its affection to the transport of goods and the spillovers, the amount raises up to 41 billion dollars.

From a traffic system standpoint, delays are a symptom of underperformance. The volume

of air traffic has been growing at a very fast pace during recent years, and it is predicted to keep growing in the forthcoming years. Assuming an increase in air travel demand following today's trend, demand will exceed the current airspace capacity before 2030 [12]. Moreover, it is a well-documented fact that delay increases nonlinearly as demand approaches capacity in the system [2]. Solutions to efficiently utilize airspace capacity while reducing time delay become critical in this scenario. Better planning for operations can help achieving these goals.

Most of the delays are caused by adverse meteorological conditions. Studies based on data from the FAA report [9] that over 70% of the air traffic delays are attributable to convective weather; therefore, a key aspect for successful planning is to develop approaches to cope with this hurdle.

Given the uncertain nature of long term weather forecasts and the factors determining other contingencies, the Air Traffic Control (ATC) efforts have leaned towards the development of initatives for a short time horizon, what is known as *tactical* planning (0-2h), since information is then more accurate.Tactical plannning falls short of achieving a general reduction of the delays since it acts in brief periods of time and in a more reactive than proactive fashion, as opposed to *strategic* planning (2-8h). On top of that, current Air Traffic Management (ATM) system is the consequence of successive additions to the first existing independent tools when commercial aviation appeared, resulting in a strongly decentralized system, focused in solving issues on localized regions of the airspace without the coordination between these tools needed to reach global optima.

In this context, an integrated tool for centralized strategic planning, which sets a first global optimal distribution with a coarse degree of detail, would be a great aid for the ATC centers.

1.2 Aim of the Project

The goal of this project is the development of an integrated ATM planning tool for the strategic horizon to help the ATC system setting, in a first instance, the guidelines for traffic management, in such a way that traffic flow in the airspace optimizes globally the total time delay.

Concretely, the project includes the formulation of the air traffic model, the construction of algorithms and methods to process raw data into entities manipulable for the model, and the design of an optimization program for the traffic flow. The result of the model are the instructions to bear to the ATC layer. These instructions must align with the existing tools on the current ATC system, i.e. prescribe control actions for which it is already prepared. For that purpose, the control actions considered are ground holding and rerouting, which are already deployed by the ATC centers.

In order to create a potentially useful tool, there are two undeniable properties the model should possess: computational tractability for large datasets, and robustness against contingencies. The former is enforced by the size of air traffic on the NAS, and the latter because the helpfulness of the tool will be proportional to its ability to deal with contingencies; conventional methods do not have much issues in clear scenarios. These are the qualities pursued and the cornerstones of the model.

This project is not a standalone and builds off over previous research on the topic. Synthesizing and using some concepts introduced later on, we can say that both Eulerian Flow models and Aggregate Route Models are not new by its own; the key distinctive trait of the work is the creation of a network redesign feature for an Aggregate Route Model in an Eulerian Flow framework, strikingly increasing model's flexibility to circumvent contingencies arisen from weather fronts. As far as the author's knowledge goes, this is the first work of this kind on the benchmark cited.

It is assumed throughout the thesis that the reader is familiar with the elementary concepts of control theory, graph theory and optimization techniques.

1.3 Definition of the Problem

The model must be capable to, from the following input data

- An initial flight plan (scheduled departures on each airport and destination airport for these departures) on a determined region, which can go from a single Air Sector to the whole NAS.
- A timespan ranging within the aforementioned strategic horizon (2-8h).
- The expected layout of the region of interest at the starting time (expected airborne flights).
- Weather forecasts and/or related weather products (see Chapter 2 for further explanation of the former).

output the control actions every airport tower needs to take, in terms of ground holdings and reroutes, respect to the initial flight plan filed during the timespan specified.

1.3.1 Aggregate route model with Eulerian flow for strategic planning

Eulerian traffic flow

Before introducing any predictive tool, we need to set a framework to study air traffic flow. *Eulerian* flow is a term coined as an analogy to the hydrodynamics field, and it refers to models controlling cells of the airspace instead of single aircraft trajectories, what *Lagrangian* models do. Intuitively, the elementary unit of a Lagrangian model would be an aircraft state (position and velocity), while for the Eulerian model it would be a delimited area (or volume) of the airspace. Eulerian models have received the attention of the research community during the last decade [24], [7], [19].

Eulerian models have two main advantages over Lagrangian methods. First of all, they are computationally tractable for a high number of aircraft, since their complexity scales with the area of the region of interest and not with the number of flights to distribute. Secondly, their theoretic properties enable the use of standard control theory and optimization methods to analyze them. The feature of Eulerian models, a coarser degree of resolution, is not a big impediment for strategic planning purposes, since it is meant to be a first approximation.

Aggregate route model

After knowing the traffic flow framework of study, we need a predictive model for the occupancy of each cell element under study. That is, given the flight demand filed for our target horizon, how will the aircraft distribute on the airspace, i.e., which zones will they occupy during their trip?

This is not a trivial task, specially if we aim to reduce the problem complexity. A possible

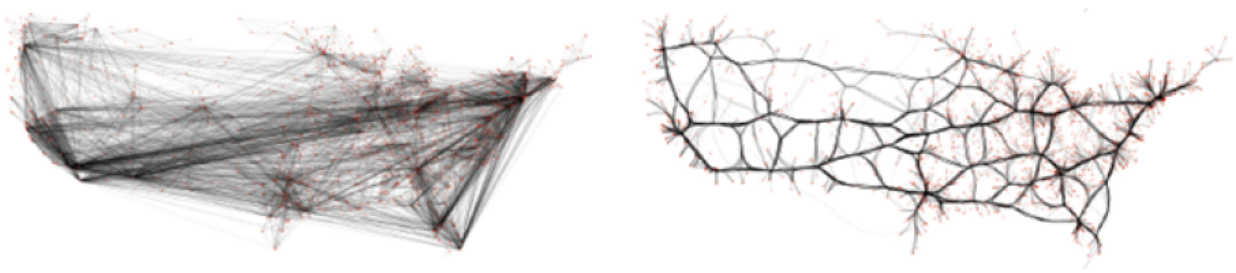


Figure 1.1: ARM generation using historical flights data, from [6]. Left, aircraft trajectories recorded. On the right, resulting routes after the clustering process.

approach is to build a simplified version of all the possible trajectories, a network of routes, akin to a roadmap. One could use the National Playbook routes; however, it is important to be sure that the routes on this network really reflect the actual paths taken in practice. The ARM responds to this wish: a network of routes is created via aggregation of historical flight paths in representative groups. Using a metric of choice, in this work the Fréchet distance, flight trajectories are clustered into main routes. The detailed procedure for generating the network of routes and a deep explanation of how it works falls out of the scope of this thesis, and can be found on [4]. A visual example of an ARM generation is shown in Fig. 1.1

Previous work on aggregate route models with Eulerian flow for strategic planning

A similar simplified version of the model proposed is outlined in [3]. The work presented on the thesis can be understood as an evolution from this model. It is also the model against which the simulations results are compared. Hence, it is a highly recommended reading.

Subsection 1.3.2 analyzes the shortcomings of the model in [3] that inspired the thesis work.

1.3.2 Adding flexibility to the ARM for weather contingencies. Overview of the solution proposed

When building an ARM, even if carefully sampling the historical data so that it includes representation of different weather contingencies, it is improbable to seize all the possible convective weather scenarios. This results in route networks that are inflexible backbones, and even if displaying multiple routes connecting O-D pairs, which apparently should offer alternatives for different scenarios, they usually represent, in the best of the cases, really suboptimal alternatives.

Two typical situations arise from the [3] type of model under weather contingencies. The first one is that all the different routes connecting an O-D pair are blocked. This can happen because there is a huge weather front, but oftentimes happens because on the ARM generation there was no relevant historical data of flights confronting similar situations; therefore, all the routes cover a fairly similar trajectory, usually around the shortest path. This is depicted on Figure 1.2 The second is that some of the routes connecting the O-D pair are available, but they imply a large detour that could otherwise be solved by slight deviations from the blocked routes. This is depicted on Figure 1.3

Because convective weather causes more than 70% of the delays, failure on effectively coping with this source of contingencies is a strong weakness for an ATM model.

The model presented on this thesis resolves this issue. When the flight plan and weather information is received, those routes of the network with scheduled flights affected by convective weather are detected. After some considerations, for those displaying characteristics indicating a redesign could be beneficial, route detours are calculated and added to the ARM. The network resulting from this addition is called ARM *augmented network*. The new routes are called *alternative routes*.

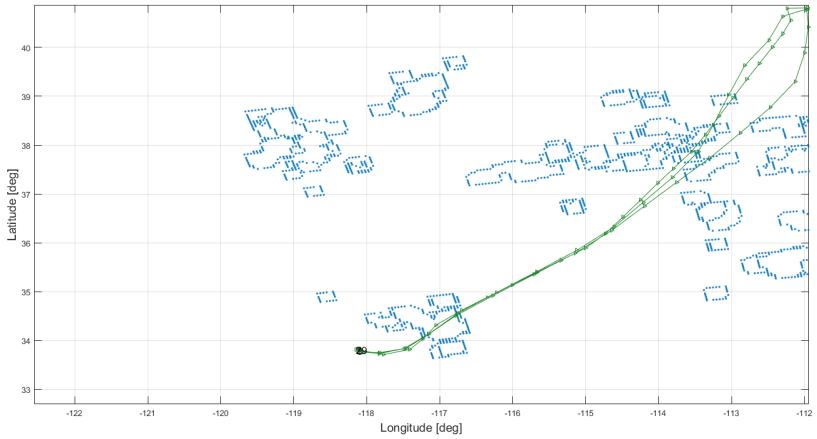


Figure 1.2: Weather blockage example for an ARM. In this case, three routes (green) are connecting the same O-D pair in the Salt Lake City Center. All of them are blocked by convective weather (blue polygons), since until the last section they follow very similar paths, leaving ground holding as the only option. However, a slight detour to the right would allow the route to work around the convective weather zone.

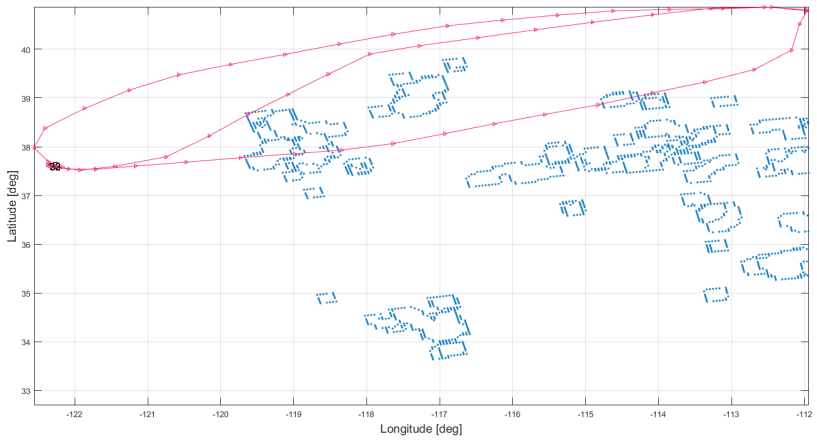


Figure 1.3: Weather blockage example for an ARM. In this case, three routes (pink) are connecting the same O-D pair in the Salt Lake City Center. Since the two lower routes are blocked by convective weather (blue polygons), the only possible choice is whether a rerouting control to the upper route or a ground holding. However, it is clear that the middle route blockage is minimal and a much more optimal strategy could come from a slight redesign of this route.

The process of redesign is performed such that the routes generated are feasible for an aircraft trajectory, but at the same time introducing minimal complexity to the initial ARM. Alternative routes can connect the original route with itself (detour), or merge with another route of the ARM. The flow to these alternative routes will be directed through an already existing rerouting control command, even though on the mathematical formulation of the model it will require some modifications.

This is the key feature and main contribution to the research community of this work. To the best of author's knowledge, as aforementioned, no other work to date has approached the issue of ARM dynamic redesigning to face weather contingencies.

1.3.3 Mathematical formulation of the framework

The route network generated by the ARM model is described by a theoretical-graph structure. Therefore, the atoms of our Eulerian model are the graph nodes. They are the static entities of the airspace on which the analysis will focus. More concretely, if the total number of routes is N_R, it is a disjoint directed graph $\mathcal{G}(X, E)$[1] of N_R subgraphs. The fact that they are disjoint subgraphs does not imply they are unrelated, since different constraints will correlate their occupancy values. Each node on the graph, ${}^{i}x \in X$, represents a waypoint with a given latitude and longitude; each link, ${}^{i\to j}e \in E$, represents an aircraft route joining ${}^{i}x$ and ${}^{j}x$.

This graph structure is suitable for the ARM description and the network expansion during the addition of alternative routes. However, traffic flow dynamics among each route nodes on the network are more conveniently stated as a linear time-discrete system. Therefore, once the routes network is completely tuned, the problem is translated from a theoretical

[1] the more standard notation $\mathcal{G}(N, E)$ is sacrificed for the sake of notation unity along the different chapters

graph structure into a controls canonical state-space representation

$$\mathbf{x} = \mathcal{A}\mathbf{X}_0 + \mathcal{D}\mathbf{d} + \mathcal{B}\mathbf{u}$$

where the state vector $\mathbf{x}$ represent nodes occupancy and $\mathbf{u}$ is the vector of optimal controls introduced. Further explanation is granted in Chapter 5 and 6. This canonical form allows as well the statement of the traffic constraints and the formalization of the optimization program.

1.4 Outline of the Thesis

This thesis is organized in four conceptual blocks and unfolds as follows.

First, in Chapter 2, the tools and guidelines to model weather forecasts into entities tractable by our approach are expounded. This, along with an ARM, sets the foundations needed for generating the alternative routes.

The method to find new alternative routes for the ARM is developed on Chapters 3 and 4. In Chapter 3, a local extension of the graph under differential constraints is proposed, along with a tailored graph search algorithm developed in Chapter 4 to find a certain number of new alternative trajectories to add to the ARM basis.

Chapter 5 models the dynamics for the ARM augmented network, posed as a discrete linear time-invariant system, with Chapter 6 establishing the constraints over these dynamics and the formulation of the problem in the canonical form of an integer linear programming program, setting the necessary structure to solve it numerically.

On Chapter 7 the performance of the proposed method is examined through simulations based on real air traffic data for the NAS, and some concluding remarks are given in Chapter

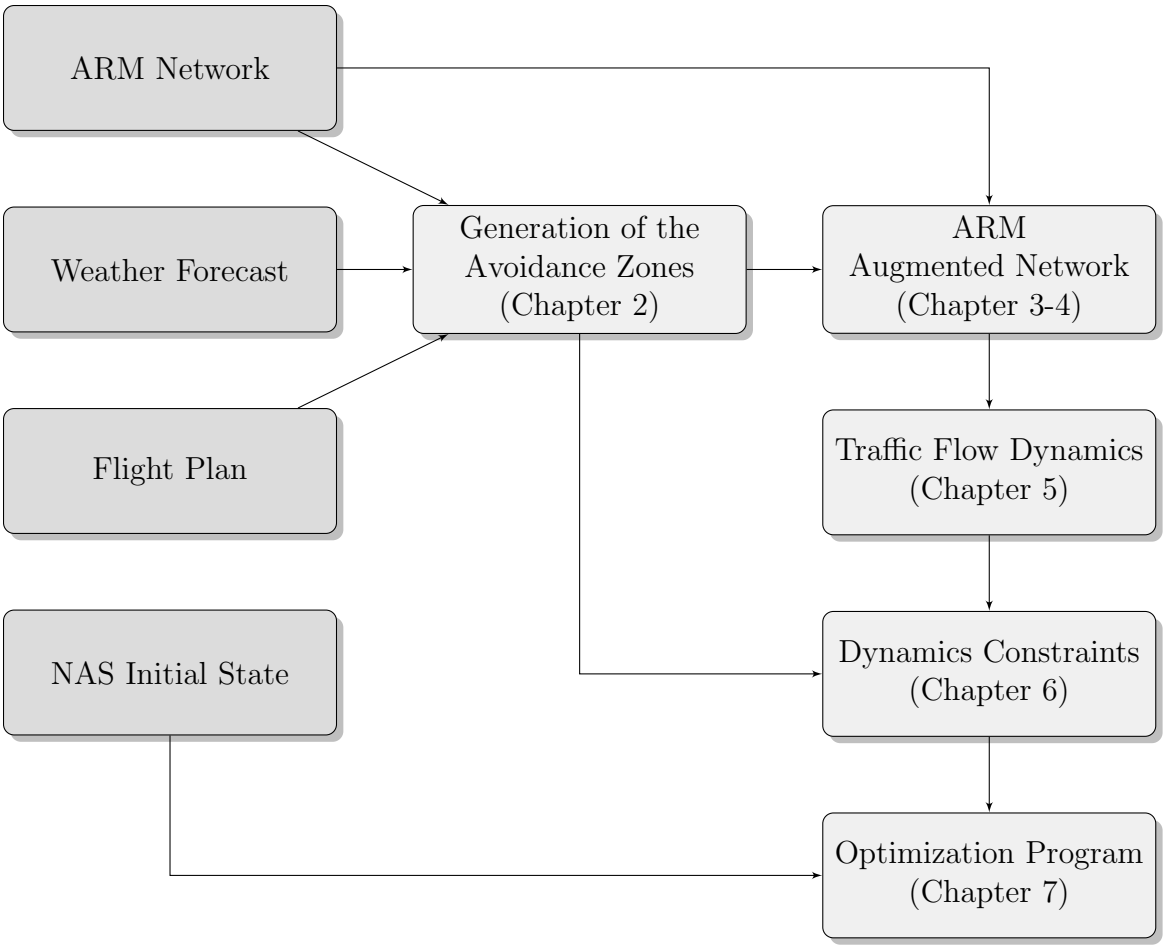


Figure 1.4: ATM Strategic Planning Tool flowchart.

8. Finally, a minimum viable case example is also attached in Appendix A, where the dynamics and constraints of Chapter 5 and 6 are worked out explicitly to help those readers who wish to get a better intuition of their nuances.

A MATLAB® implementation of the algorithms and programs discussed throughout the document can be found in `github.com/afrancir/ATMProject`. Some representative samples have been included also in Appendix B.

Chapter 2

Avoidance Zones from Weather Forecasts

The main threat and source of instability for air traffic operations is convective weather. One of the challenges in modeling convective weather for ATM operations is to understand the impact it will have on the traffic flow from just raw meteorological information. Perhaps the toughest part is to anticipate the exact deviation from the original route a weather front provokes on aircraft trajectories.

This Chapter discusses this issue, argues the approach taken, and subsequently presents the method for mapping the convective weather information into its effects on our model.

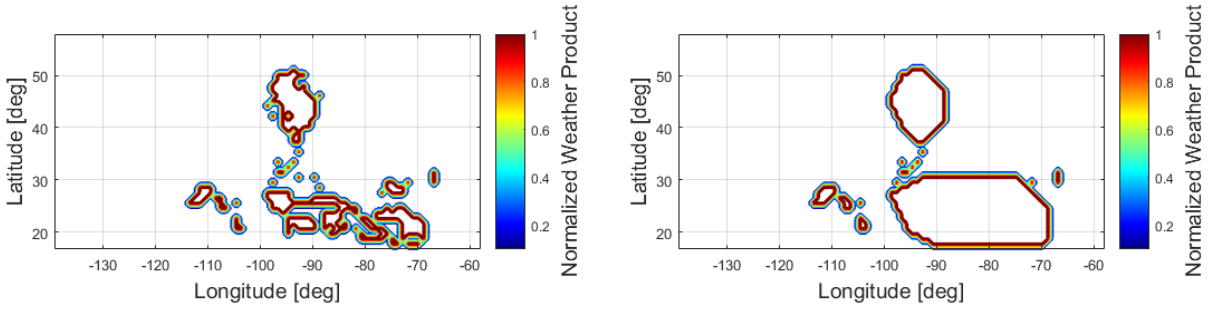
2.1 On Evaluating the Impact of Convective Weather in Air Traffic

Given the lack of reliable forecasts on the strategic horizon [23], it is common to use weather products to determine potentially problematic areas. Weather products are usually made out of a combination of physical indices, such as wind speed or humidity, but may not limit to that. Recent studies [21] have worked on including historical pilot decisions in similar circumstances to weight the odds for an area to be crossed.

Yet, none of the known products seems to succeed on pinpointing the shape of the avoidance surface emerging from the weather-affected area. While supplying information clearly related with the zone to be avoided, weather products do not embed many factors that play a role on the decision making process, such as visibility, aircraft maneuverability, etc., not to mention those more case-dependent like pilot experience or airline guidelines.

Hereafter it will be assumed an adequate weather product is provided, as well as a threshold τ over which the regions become avoidance zones. The model works with any product as far as this last piece of information is given. Regions enclosing values of the product over the threshold are considered inaccessible, i.e. this is a hard constraint. Adding fuzzy logic on the model to account for the probability of traversal by some of the aircraft can be explored in a future; however, it does not seem the best procedure to a priori assign some flights to go through dangerous areas if it can be prevented.

These zones shape the core of what avoidance zones will be. In order to account for some of the differences between these and the actual boundaries aircraft will follow, we process the contours of the regions as explained in Subsection 2.3, which at the same time eliminates the superfluous details the contour shape could initially carry on. The method developed, using a custom clustering strategy, considers the limitations imposed by aircraft maneuverability,



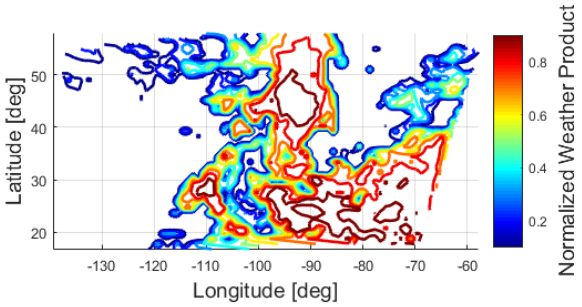
(a) Countors of zones with product value over τ (b) Processed avoidance zones

Figure 2.1: Example of the difference between the unpolished and polished avoidance polygons.

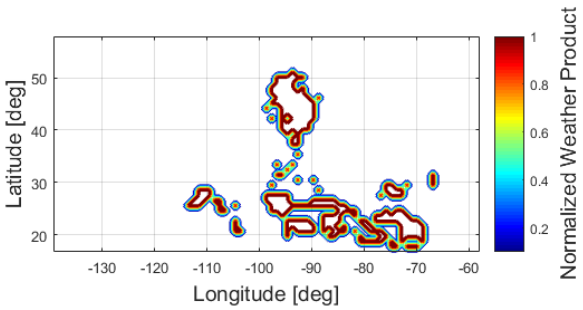
and also discards unrealistic scenarios like backtracking on the route, even if feasible.

2.2 Generating the Core of the Avoidance Polygons

Let pM be a matrix with each of its elements representing the average value of the weather product in a grid cell of the airspace region of interest, i.e. a delimited area of latitude-longitude pairs at the same altitude. We set all values greater than the threshold τ to 1 and 0 otherwise. Elements which are above, below, or to either side of another element are considered neighbors of the latter. Groups of connected neighbors form the polygons of what will be the nucleus of the avoidance zones.



(a) Weather product value contours



(b) Contours with values over τ

Figure 2.2: Avoidance polygons core generation from the weather product.

A relevant aspect is the granularity of the matrix. Excessively fine resolution will not result in any improvement on the planning if it exceeds the aircraft reaction capabilities, but will increase the matrix size quadratically. As a rule of thumb, an appropriate dimension is to discretize the space such that aircraft maneuverability permits going from a matrix element to its neighbor if its heading angle is below $\pm 90°$ relative to the line connecting both neighbors centers.

If multiple forecasts for different periods during the horizon are available, for each grid cell of the airspace region only the most pessimistic values are considered. Setting a different grid cell value for every time-step could lead to cumbersome calculations when generating

alternative paths since we do not know beforehand at which time-step every of these cell grids would potentially be crossed.

For big airspace areas, a good procedure is to break the airspace down into smaller subregions and process them independently, then run the algorithm for the whole region with the already preprocessed polygons of each subregion; this alleviates the computational load.

2.3 Avoidance Polygons Processing: Clustering the Convex Hulls

Clustering strategies have traditionally been of two types: density based clustering and hierarchical clustering. The algorithm presented is not either of these types.

Density based algorithms cannot be used on our problem since they do not preserve the original area boundaries, which means that some inaccessible regions may turn into non-affected areas after the clustering. On the other side, hierarchical clustering imposes a trade-off between fine and course resolution we are not willing to accept; we want to keep a fine level of detail there where it is needed, but without accumulating the same details in other zones where these are superfluous.

For that purpose, a custom algorithm was designed based on iterative calculations of the convex hulls of each closed avoidance polygon. Taking convex hulls of every polygon makes sense in order to avoid unrealistic backtracking in the trajectories of the detours. However, calculating the convex hull of all the polygons directly would basically merge the different isolated polygons into one big smear, creating a rough approximation, similar to hierarchical algorithms with large intercluster distances. In order to preserve spacing between isolated polygons that may be feasible to cross, the algorithm, schematized in Pseudocode 1, does

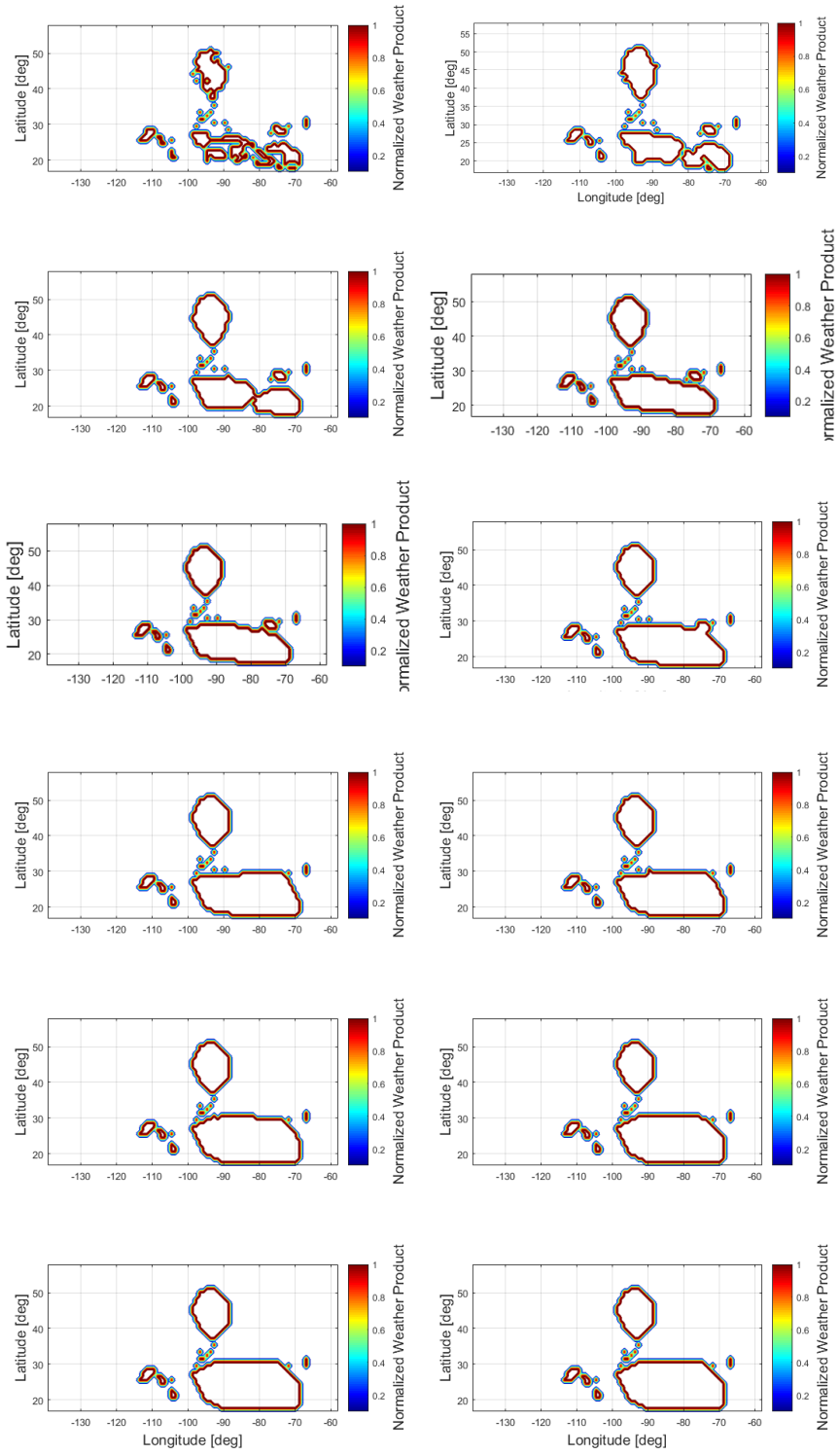


Figure 2.3: All the iterations for the polishing process of the avoidance polygons in a case test. Data used correspond to the 07/01/2011 Cloud Top Height value at 3pm UTC-7. Evolution goes from left to right, top to bottom.

the following: first it identifies the unconnected polygons of pM. After that, each of these polygons is taken separately to calculate its convex hull. The convex hulls are saved apart and finally pasted back together into a matrix. The procedure stops when new iterations do not produce any change on the avoidance polygons.

Algorithm 1 Main Avoidance Zone Generation

1: **input:** $r \times c$ binary matrix pM
2: **output:** $r \times c$ binary matrix $pM2$
3: initialize ppM to $0_{r \times c}$
4: **while** $pM \neq ppM$ **do**
5: $\quad ppM \leftarrow pM$
6: $\quad pM \leftarrow$ CHulls(PClassifier(pM))
7: **end while**
8: $pM2 \leftarrow pM$

pM can vary after the first iteration because overlaps between convex hulls are possible; since the overlap of convex hulls is not necessarily convex, the convex hull calculation process must be repeated.

A salient property of the algorithm is the absence of parameters to tune. Both density based and hierarchical algorithms require the adjustment of constants, whether for the number of clusters, the distance between centroids, etc. Here this is not necessary.

2.3.1 Independent polygons detection

To detect the distinct closed polygons forming the avoidance zones, the algorithm described on Pseudocode 2 is used. It is similar to flood-fill algorithms [18], sweeping the matrix elements' neighbors in order to see if they are connected, and assigning different labels to every unconnected polygon. It is noteworthy that it only needs to sweep the matrix in one direction to achieve that.

Algorithm 2 Closed Polygon Identifier

```
1: function PCLASSIFIER(pM)
2:     input: r × c binary matrix pM
3:     return: r × c integer matrix pMC
4:     initialize pMC to 0_{r×c}
5:     initialize classCounter to 0
6:     for i = 1 to r do
7:         for j = 1 to c do
8:             if pM(i, j) = 1 then
9:                 initialize top, left to 0
10:                switch upon top, left values do
11:                    case top = 0, left = 0
12:                        increment classCounter
13:                        pMC(i, j) ← classCounter
14:                    case top = 1, left = 0
15:                        read topC, the class of top, from pMC
16:                        pMC(i, j) ← topC
17:                    case top = 0, left = 1
18:                        read leftC, the class of left, from pMC
19:                        pMC(i, j) ← leftC
20:                    case top = 1, left = 1
21:                        read leftC and topC from pMC
22:                        if topC, leftC, are equal then
23:                            pMC(i, j) ← leftC = topC
24:                        else
25:                            minC ← min (topC, leftC)
26:                            initialize upS, lS to 0
27:                            while topC, leftC, and minC not equal do
28:                                switch arg max(topC, leftC, minC) do
29:                                    case 1
30:                                        increment upS
31:                                    case 2
32:                                        increment lS
33:                                    case 3
34:                                        break
35:                                end switch
36:                                topC ← pMC(i − (1 + upS), j − lS)
37:                                leftC ← pMC(i − upS, j − (1 + lS))
38:                                pMC(i − upS, j − lS) ← minC
39:                            end while
40:                        end if
41:                end switch
42:            end if
43:        end for
44:    end for
45: end function
```

2.3.2 Convex hull calculation

Having every independent polygon labeled, each one of them is taken apart to calculate its convex hull. For calculating the convex hull we can use a gift wrapping algorithm [13] or any equivalent technique. Once the convex hulls of all the independent polygons have been calculated, the hulls are pasted back onto the original plane to check for overlaps between them. The structure of the function is attached in Pseudocode 3.

Algorithm 3 Isolated Convex Hull Calculator

```
1: function CHULLS(pM)
2:     input: r × c integer matrix pC
3:     return: r × c binary matrix pM
4:     Id ← set of unique pC values
5:     Initialize pM to 0_{r×c}
6:     for i in Id do
7:         Initialize pCI to 0_{r×c}
8:         pC elements with value i are set to 1 on pCI
9:         calculate pCI convex hull, values inside the convex hull are set to 1 as well
10:        pM ← or(pCI,pM)
11:    end for
12: end function
```

2.4 From Avoidance Polygons to the No-Fly Zones

The final avoidance polygons will be used as a constraint for the traffic flow dynamics. Any node within the boundaries of the avoidance polygons' coordinates will have a null occupancy limit, or in other words will not be able to host any aircraft. This is known in the aeronautical field as a No-Fly Zone.

It is important to stress however that not all the No-Fly Zones introduced as constraints in the planning problem are due to convective weather. Other reasons may deem a region a No-Fly Zone, such as prohibited airspace due to military concerns. In other words, all the weather avoidance polygons turn into No-Fly Zones, but not all the No-Fly Zones come from

weather avoidance polygons.

Chapter 3

Graph Extension: Generating Feasible Alternative Routes

Obtaining avoidance polygons from meteorological information translated our data of interest into entities tractable in a path planning problem. In the following chapter we tackle the path planning problem itself, setting the basis for how the alternative routes will be produced. For this reason, the generation of the graph extension is a cornerstone in our work.

There are many existing approaches for trajectory calculation in the presence of obstacles, such as Artificial Potential Fields or more generally any PDE-based model, that require vast computational efforts to create very precise trajectories. For the purpose of creating a coarse approximation of an alternative route these are clearly unnecessary. Graph-based methods seem a priori more adequate to deal with an ARM reroute generation, particularly since the routes network is already a digraph itself. Still, they cast two big challenges.

The first is to generate alternative routes that, while simplified, are feasible to cruise by the aircraft; our problem is limited by the motion constraints of an airplane, or in other words, by its dynamics. The path search on the state space is not free but instead constrained to

certain equations of motion.

The second is how to create a graph extension big and complex enough to find the best rerouting possibilities while keeping a low dimensionality. We want to avoid the situation of [22] where an extensive optimization search is required. Our goal is not to find the most optimal path from start to end every time a flight is scheduled between two airports. This, besides potentially sterile in strategic planning contexts where forecasts are highly uncertain and subject to change, is extremely computationally intensive. Our intention is to add flexibility to the route network generated via ARM upon weather contingencies blocking multiple routes, while keeping the characteristic low dimensionality of our network.

Hence, the target set when generating the alternative routes to circumvent weather events is to create alternative traversable routes which introduce the minimum number of new nodes to the network. This is indirectly related to a time delay minimization (shortest detours will be prioritized versus longer ones) but not exactly equivalent. It is important to keep this in mind since the method to determine goal nodes on the extended graph is founded on this idea.

We will give the procedure to generate a set of new graph branches rooted on the blocked nominal route, which will be referred as "the graph extension", and the specific details on how to produce feasible nodes and edges, selecting the start node to root the graph, and delimit the extension of this graph. Alternative routes, if possible, will arise from solving the shortest path problem on this extended graph.

3.1 Constraints for the Graph Extension

3.1.1 Kinodynamic constraints

For the aircraft path planning problem, simplified 2D motion dynamics are considered, such that the generated trajectories will belong to the same plane the ARM routes do. Edges connecting the nodes, besides abstract connectors, are also in our problem the physical path the aircraft will follow; thus, it is important to recreate the edges curve accurately rather than simply generating them in any geometry.

We aim for the simplest acceptable dynamics model to recreate realistic aircraft trajectories. Loss of performance in finding the best alternative route because of neglecting the subtleties of piloting is fine for our purposes; to set a coarse route distribution optimization, a high degree of detail on the trajectory calculations is undesired. What we want is to prevent the calculation of unfeasible routes that aircraft are not capable to follow due to their motion constraints.

The aircraft is modeled as a point agent, with constant speed v (already fixed by the ARM choice for this specific route), which moves in a circular motion in any curvature over the minimum turning radius R threshold. The minimum turning radius is obtained by $R = \frac{v^2}{g \tan(\theta)}$, where θ is the maximum banking angle for the aircraft and g the g-force on the current elevation. These values are considered constants, and nuances related to their variations due to true airspeed and other conditions, ignored. The node's spacing must also be the same than the used on the ARM for the route, $d = v\Delta t$, where Δt is the time step for our discrete dynamics model, already fixed also by the ARM choice. Given R and d we can obtain the maximum turn angle achievable by the aircraft in this distance with $\Delta\alpha = \frac{d}{R}$. The parameters required for the dynamics model are summarized on Table 3.1.

Parameters	
v	cruise speed
g	g force (altitude)
θ	banking radius
Δt	time-step

Table 3.1: Parameters needed for the Dynamics model generation

3.1.2 Other constraints imposed

There are restrictions imposed on the graph generation further than the strictly physical. One relates to a trait we already introduced on Chapter 2 for polishing the avoidance polygons, and that is to avoid backtracking. We impose a 90° difference limit respect to the nominal route segment heading; edges generated with heading angles over this threshold will be pruned.

The other, more involved, relates to the second big challenge mentioned during the chapter introduction: bounding the extension of the path planning search. The generation of the graph grows exponentially with the number of levels allowed, thus it is fundamental to enclose the new graph in a limited surface. This will also preempt feasible but nonsensical routes: if there is a reroute to go from Houston to Austin passing through Dakota, even if dynamically feasible, we do not want to consider it.

Pinpointing the concrete value for these boundaries is more an art than a science, since it comes down to a trade-off rather than to any inherent limitation on how the reroutes can be. Our method takes the diameter ϕ of the largest avoidance polygon blocking the route, and centers a square at the median of the set of consecutive blocked points, rotated 45 degrees from the blocked sector direction. [1]

[1] On the example of Fig. 3.1 can be observed that the restriction acts over the nodes positions, but some of the edges go slightly over the limits. This responds to the fact that, being this a soft constraint with an arbitrary value to simply avoid huge searches, enforcing the condition to edges intersections would make the check unnecessarily expensive, while for nodes the algorithm is $\mathcal{O}(1)$.

If more than one sector of the route is blocked, the distance between the two blockages, l_1, is compared to both diameters ϕ_1, ϕ_2; if any of them is larger than l_1 the two sets of blocked nodes plus the intermediate ones are all merged into a unique set of consecutive blocked nodes, then the boundaries set with a diagonal of length $2(\phi_1 + \phi_2)$. The procedure can be repeated in a bottom-up approach in case there are many isolated blockages on the same route.

Similar solutions have been proposed on related works. In [21] the boundary contour chosen is an ellipse of size proportional to the blocked section length. A more sophisticated method could be interesting for further research, but is out of the scope of this work.

3.2 Recursive Procedure for the Graph Extension

Let the actions set be $\mathcal{U} = \{\text{turn left}, \text{straight}, \text{turn right}\}$ and the corresponding turning radii for these actions $\mathcal{R} = \{-\Delta\alpha, \infty, \Delta\alpha\}$, with the minus sign simply indicating the side of the plane the turn leans into. In order to span as much as possible, the minimum turning radius has been chosen for the turn, but any $|R| \geq \Delta\alpha$ would work; the same can be said about the number of actions: more actions can be added to the set as desired, the selected set is intended to cover the basic motions according to the dynamics simplicity we pursue.

The graph extension is built by recursively expanding every node taking all possible actions. Starting from the start node, each of the possible actions is taken to generate a new node and an edge connecting them. After the generation of a new node, it is checked if it violates any of the constraints; in case it does, the leaf is pruned. Otherwise, the same process starts over again from this new node. If the node complies with the goal conditions, the expansion is terminated but the leaf is not pruned.

Therefore, with the $\mathcal{U}$ proposed, every node will generate a maximum of three leaves hanging

from it. The procedure ends when all the leaf nodes can not generate children without violating the constraints or are already a goal. A simple example of the generation process is attached on Figures 3.1 and 3.2.

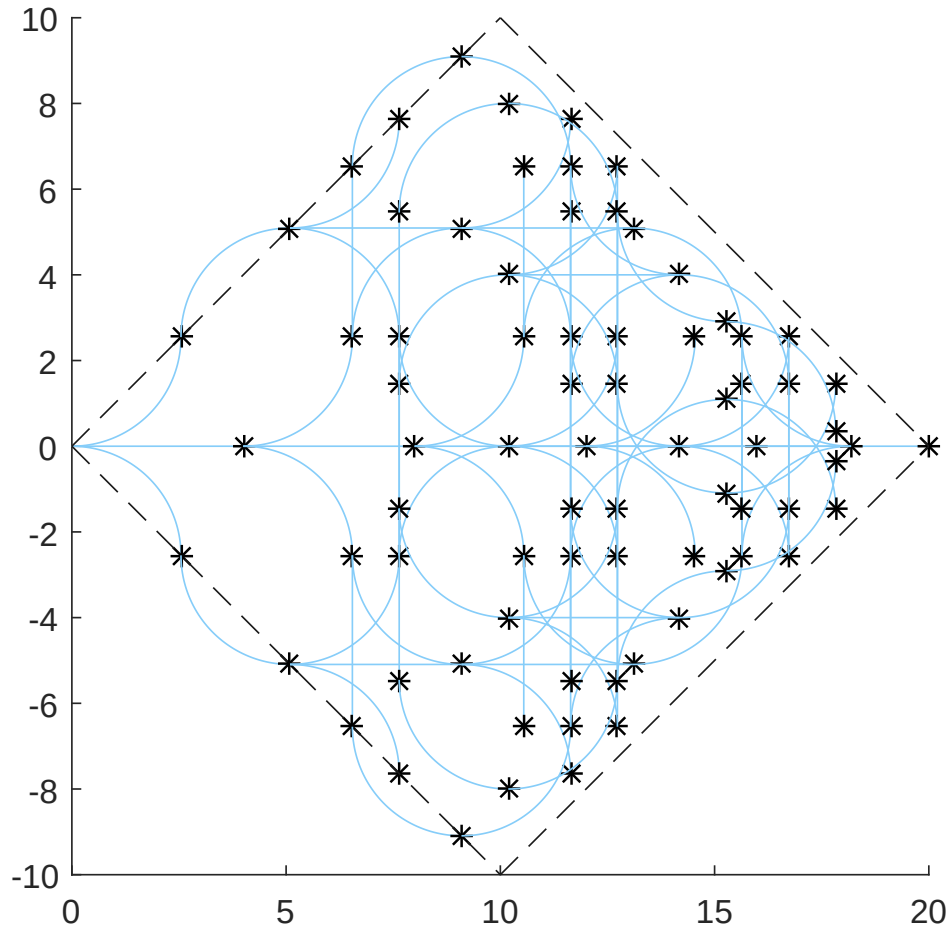


Figure 3.1: Graph Extension generation example. In dashed lines, the surface constraints imposed. The expansion starts at $(0, 0)$ with a $0°$ heading. At every new leaf three children nodes are generated: one after turning left in a circular motion with radius R for a distance d, another one after the symmetric turn to the right, and a last one after a straight path of distance d. These children are evaluated to check if they comply with the position and angle restrictions; if they do, they are inserted to a stack to be further expanded. The process is over once there are no remaining nodes to expand. Parameters values for the example are $R = 2$, $v = 2$, $dt = 2$.

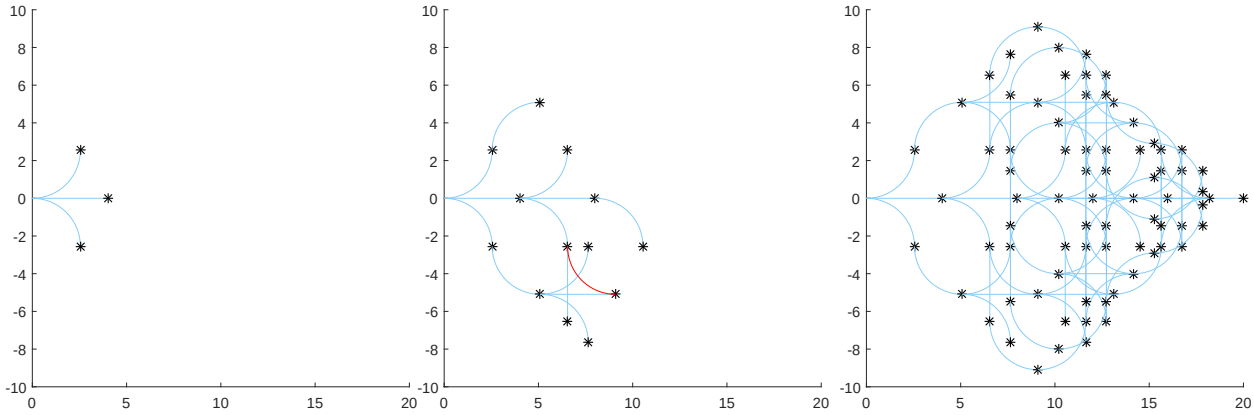


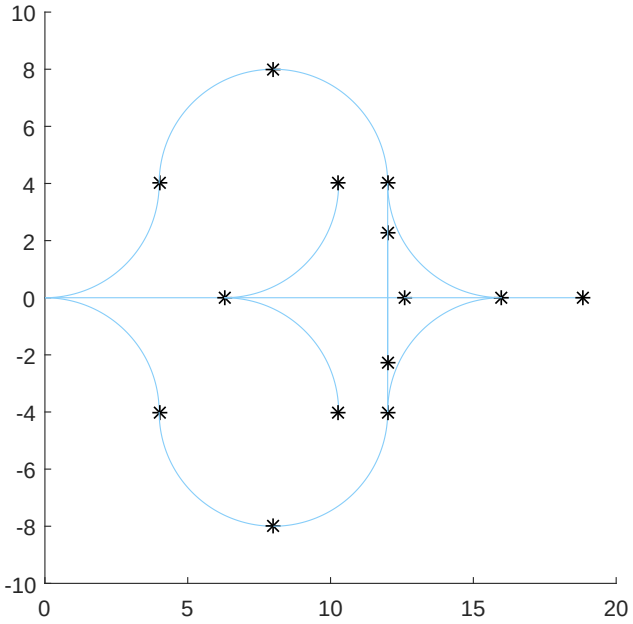
Figure 3.2: Multiple stages of the Graph expansion on Figure 3.1. On the left, the first leaf of nodes. In the middle, in red, after some leaves expansions, an edge resulting from the propagation of a node leads to an already existing node, showing why the resulting structure is a graph instead of a tree. On the right, full graph expanded: there are no remaining nodes which could generate children without violating the restrictions imposed for position or heading.

3.2.1 Termination condition: determining goal nodes

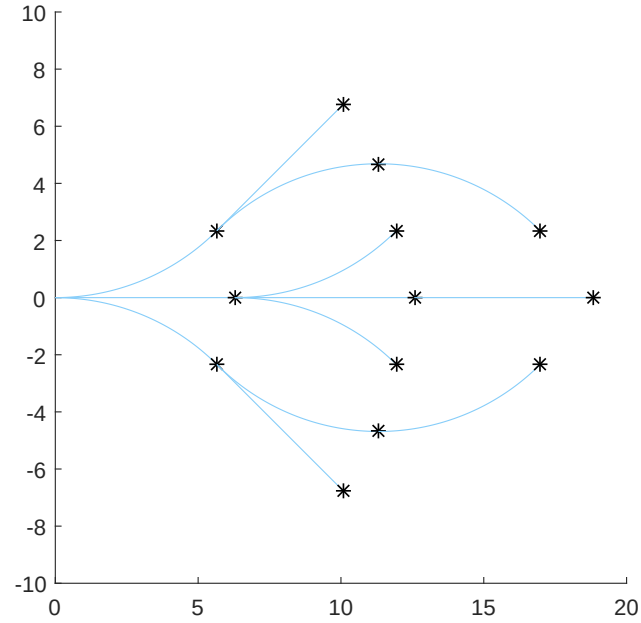
A node whose edge crosses any of the nominal route segments after the blocking zone, or any non-blocked neighboring nominal route, with an angle $\leq \Delta\alpha$ respect to the crossed segment, becomes a goal node.

3.2.2 Iterative approach for selecting the start node

The start node is unique. To decide which is the most appropriate node of the nominal route to start from and avoid excessively large graph extensions, the last non-blocked node is taken at first, and the graph extension generated from there. If the resulting graph does not have a goal node, the procedure is repeated starting from the predecessor node from the nominal route. The iterations continue until the graph generated contains goal nodes or the departing node of the nominal route is reached. An example of this procedure is depicted in Figure 3.1.



(a) Example from figure 3.1 with larger timesteps (coarser exploration). Parameter values are $R = 2$, $v = 2$, $dt = 4$.



(b) Example from figure 3.1 with larger timesteps and curvature radius (less maneuverability). Parameter values are $R = 4$, $v = 2$, $dt = 4$.

Figure 3.3: Different graph layouts by tuning the dynamics parameters.

If future updates block the generated network such that it does not reach any goal node, the network extension procedure is restarted.

3.3 Spatio-Temporal Graphs for Time-Varying Edge Costs

Succinctly, since all the nodes will have the same time-distance among them, the default cost of every edge will be set to the same constant value $c = \frac{v}{\Delta t}$, except for those crossing avoidance zones, which are set to $c = \infty$.

However there might be cases where a node is unreachable at the beginning of the time horizon and clear out after a few time-steps, or vice versa. In other words, an edge's cost will vary depending on the moment it is traversed. This graphs are known as spatio-temporal graphs.

For example, if our paths gets to ${}^{i}x$ via ${}^{j}x$ in the time interval going from t_0 to t_1, the cost to consider is ${}^{i \to j}e_0$; if it does the same path during the interval going from t_k to t_{k+1}, the cost to evaluate then is ${}^{i \to j}e_k$.

To account for this fact, the graph nodes are cloned in N_t layers, and edges connect each one of these layers with their value at every time-step transition. Figure 3.6 can help to visualize this. An interesting property is that even if the number of hard memory addresses required to store the nodes is multiplied by N_t, the computational time, usually the bottleneck for our purposes, is unaltered. This happens because the number of total edges remains the same and a graph search depends on it, not on the nodes.

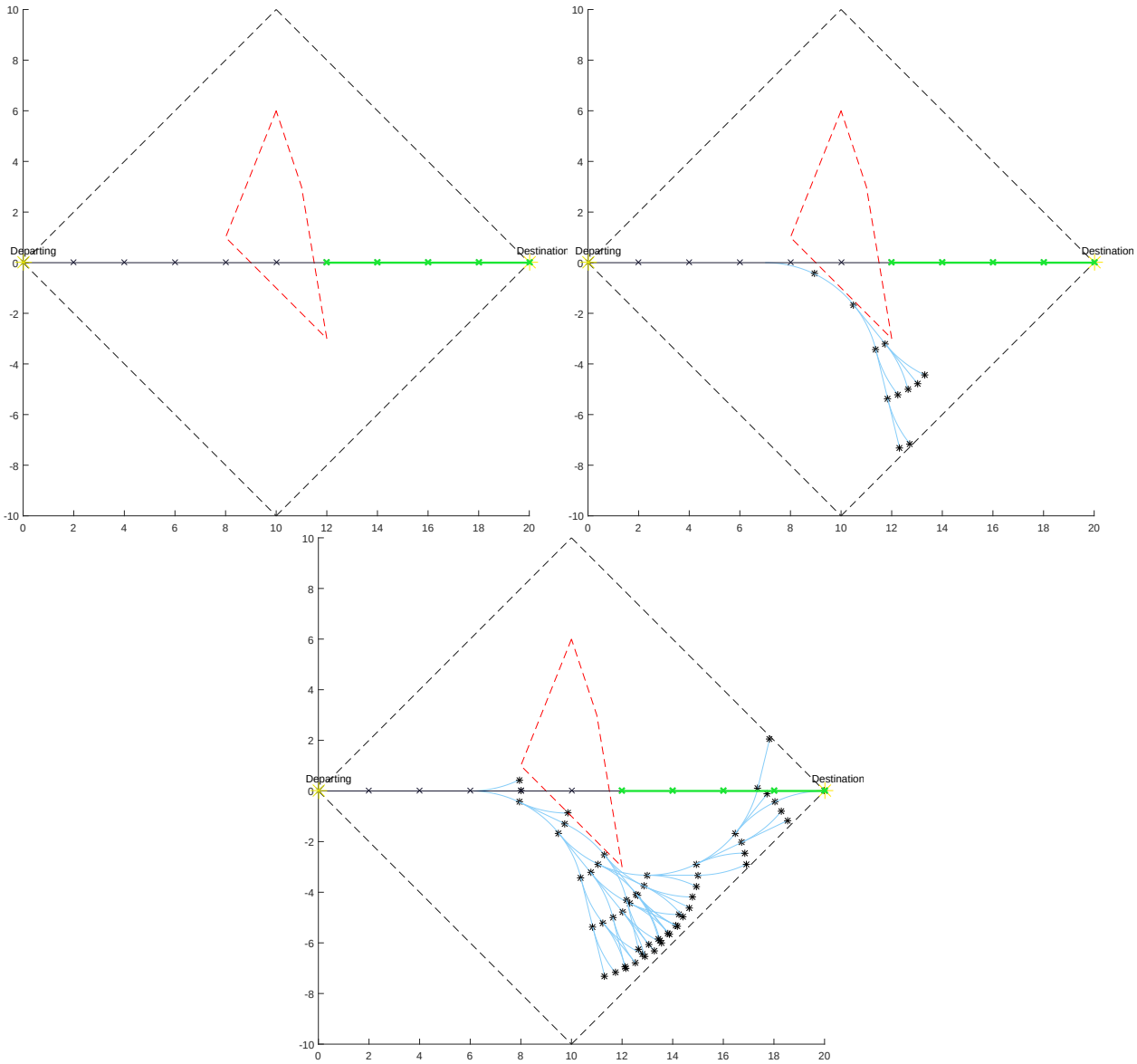


Figure 3.4: Basic example showing the recursive approach for graph extension to find an alternative route. Route origin at $(0, 0)$ with destination $(20, 0)$, all segments aligned at $0°$, with a time-step between nodes of $\Delta t = 1$. The aircraft dynamics values are a radius of curvature $R = 4.5$ and cruise speed $v = 2$. An avoidance zone (which would be the resultant aggregation of all the time-span avoidance zones for the region) depicted as a red dashed contour, blocks ${}^{6}x$. Position limits are represented by the dashed, black contour. The first non-blocked node is ${}^{7}x$, therefore all edges and nodes coming after ($\{{}^{8}x, ...{}^{11}x\}$ and $\{{}^{7\to 8}e, ...{}^{10\to 11}e\}$) , are part of the termination condition to set goals and for this reason are highlighted in green. On the upper left figure image, the problem described. On the upper right, the expansion of the graph is performed starting from the last non-blocked node, ${}^{5}x$, at $(8, 0)$. When the expansion is terminated no goal node has been generated; in consequence, the expansion starts over again from the precedent node, ${}^{4}x$, at $(6, 0)$, this time generating branches reaching at least once the termination condition, what can be seen in the lower figure.

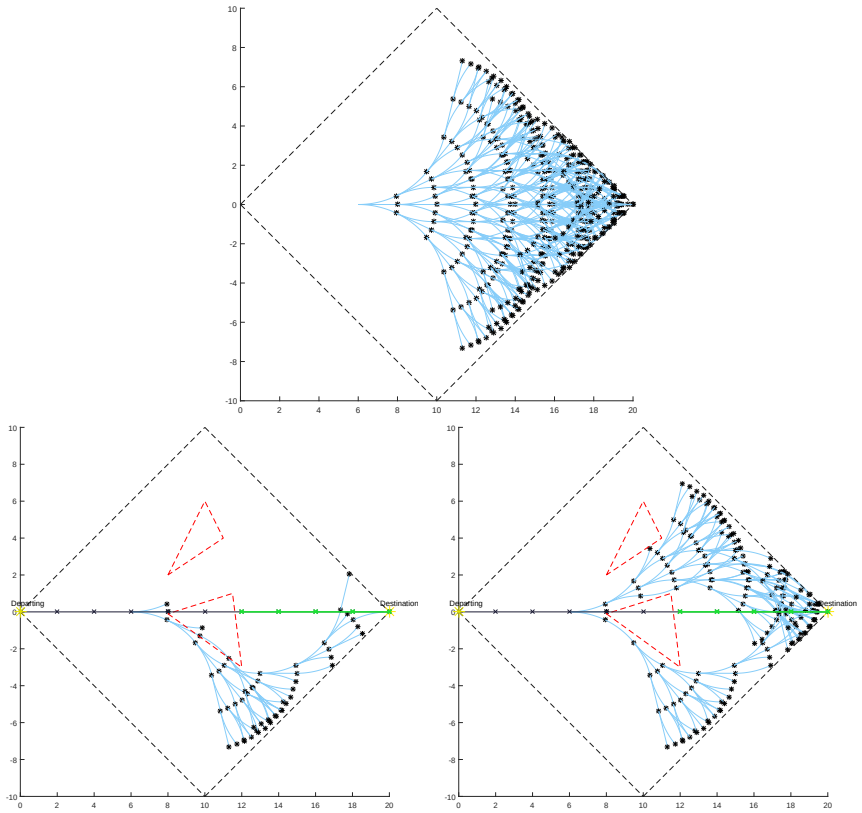


Figure 3.5: Continuation of the problem in Fig. 3.4 . On the upper figure, entire generation of the graph from the first node that lead to valid results (${}^4x$) without the avoidance zones blockage pruning. In the lower two figures there is an example of why this is necessary. If only the graph generated with the pruning on the avoidance zones was kept, this could restrict the search to suboptimal paths in the future when receiving updated forecasts for the avoidance zones' shape. On the lower left figure, the case without having generated the entire unblocked graph after an avoidance zone update. On the right, a visual recreation of all the nodes that will be available if the whole graph is generated and edges costs updated instead, allowing to take advantage of the paths crossing the new opening between both avoidance zones to find a faster reroute (∞-valued branches have been removed for visual clarity even though they would remain there).

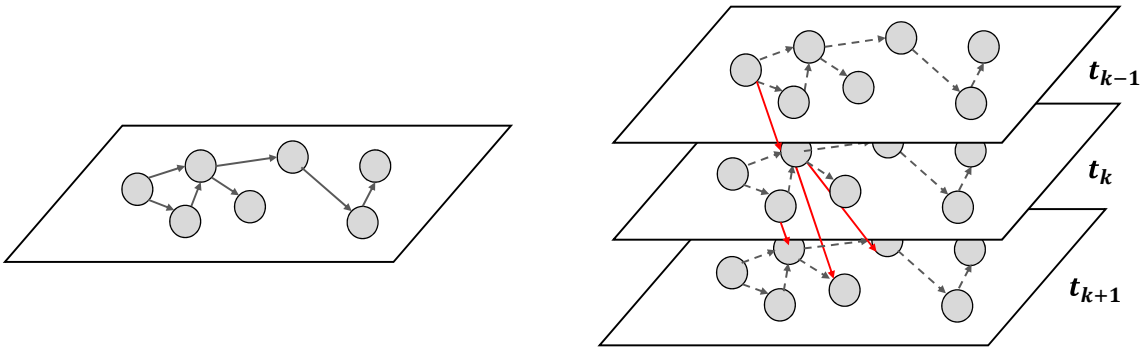


Figure 3.6: Spatial-Temporal graph representation. While the number of nodes augment is proportional to the total number of time-steps allowed, the number of edges remains unaltered.

Chapter 4

Extended Graph Search: Alternative Routes Selection

We introduce here the algorithm used to find the minimum cost path in the Extended Graph. The solution to the minimum cost problem will become the alternative route of choice, adding the sequence of nodes as a new route of the network.

The algorithm is a rehashing of Lifelong Planning A^* (LPA*) with multiple goals. Albeit using A* for ATC-related applications has been documented in previous works [22] [19], to the best of the author's knowledge this is the first time LPA* has been used in this context.

4.1 Graph Search Algorithm: Lifelong Planning A^*

4.1.1 Multiplicity of goals

The only modification on the introduction of multiple goals on the respective demonstration is indirect through the heuristics consistency; therefore we embed the multiple goals feature

as part of the definition for our heuristics, and demonstrating its consistency along with the theorems on [14] the completeness and correction of the algorithm are guaranteed.

4.1.2 Heuristics

Heuristic searches use qualitative information in the form of estimations of the goal distances to project the search to the more promising regions, solving the shortest path problem faster than uninformed methods.

Instead of expanding nodes with minimum cost $g(x)$, as we would do for a minimum cost expansion algorithm (Dijkstra), the addition of a heuristic function $h(x)$ prioritizes the expansion of nodes with lower cost but whose heuristic value also indicates they are closer to a goal. Analytically, they expand nodes regarding their $f(x)$ values

$$f(x) = g(x) + h(x) \tag{4.1}$$

For the heuristic function we chose the orthodromic distance Ω (great-circle distance) to the closest goal node x^g. Let $X^g({}^i x)$ be the set of reachable goal nodes from ${}^i x$, then our heuristic is formally defined as

$$h({}^i x) = \min_j \Omega({}^i x, {}^j x) \quad \forall\, {}^j x \in X^g({}^i x) \tag{4.2}$$

In order to ensure correctness and completeness of the algorithm, the heuristic must possess certain properties: it has to be positive semi-definite, a condition always accomplished for a distance, and *consistent*, which roughly means $h(x)$ must be an underestimate of the actual distance to the goal. A formal proof of the heuristic's consistency for our candidate function is outlined in Section 4.2.

4.1.3 Incremental search

The striking decrease of the computational times granted by LPA* respect to A* (around two orders of magnitude [14] for certain cases) belies on its incremental search structure. Incremental approaches reuse knowledge gathered on previous searches to find solutions for similar tasks, achieving this way greater speeds, specially on those situations where changes of the layout respect previous searches are small.

Eq. 4.3 shows the consistency function used to identify those regions of the graph that indeed require a re-exploration (without the need of traversing the graph again)

$$g^*(x) = \begin{cases} 0 & \text{if} \quad x = x^{start} \\ \min_{x' \in Pred(x)} \left(g^*(x') + c(x, x') \right) & \text{otherwise} \end{cases} \tag{4.3}$$

4.2 Completeness and Correctness of the Algorithm

Completeness and correctness proofs for LPA* are already presented on [15]. Multiplicity of goals does not alter any of the [15] theorem's conditions as long as it can be guaranteed that $h(x)$ is consistent. Hence, if we prove the consistency of the heuristics it will be enough to ensure that the algorithm will always find a solution and this will be optimal.

Consistency is defined as

$$h(x) \leq c(x, e, x') + h(x') \tag{4.4}$$

where $h(x)$ is the value of heuristic for node x, and $c(x, e, x')$ the cost to traverse arc e connecting x with x'.

By definition, the shortest distance between two points on a spherical surface is the orthodromic distance. Therefore, adding an intermediate point the two will increase the distance unless the intermediate point falls on the same great circle, i.e. $\Omega(x, x^g) \leq \Omega(x, x') + \Omega(x', x^g)$. Using this inequality:

$$\begin{aligned} h(x_i) &= \min_j \Omega(x_i, {}^j x) \qquad \forall\, {}^j x \in X^g \\ &\leq \Omega(x_i, x_{i+1}) + \min_j \Omega(x_{i+1}, {}^j x) \\ &\leq \Omega(x_i, x_{i+1}) + \min_{j'} \Omega(x_{i+1}, {}^{j'} x) \\ &= \frac{v}{\Delta t} + \min_{j'} \Omega(x_{i+1}, {}^{j'} x) \\ &= c(x, e, x_{i+1}) + \min_{j'} \Omega(x_{i+1}, {}^{j'} x) \\ &= c(x, e, x_{i+1}) + h(x_{i+1}) \end{aligned} \tag{4.5}$$

Hence the heuristic is consistent. The second inequality arises from the fact that the reachable goals set may differ between x_i and x_{i+1}.

Chapter 5

Route Network Model as a Linear Time-Invariant System

In this chapter we translate the graph structure used to describe the routes network into a time-discrete LTI system. Expressing the network dynamics in this form allows to tackle the problem from a classic control theoretic perspective and use the optimization techniques developed in this context, as explained on the next chapter.

Each component of the state vector is the number of aircraft associated with a particular node. To account for the controllability differences existing among different routes, some convenient distinctions are introduced. The network is divided into four components: internal, entering, exiting, and overflights. Internal flights occur inside the NAS limits from takeoff to landing. Entering flights depart from outside the NAS but at some point its route enters it and lands on one of the national airports; exiting routes constitute the inverse case. Overflights are flights that neither depart nor land in any of the national airports, but whose route crosses over the national airspace at some point.

We will refer to the node's type of route by a right superindex. For example, node 13, part

of an internal route, at time t_4, would be written as ${}^{13}x^{in}_{t_4}$. For rhythm purposes, particular examples are minimized in this section. Nevertheless, the author encourages to intermingle the reading of this chapter with Appendix A in order to gain intuition of the form the expressions introduced take in specific cases.

5.1 Air Traffic Flow Uncontrolled Dynamics for a Network Component

Since the dynamics for every component are the same, we will introduce them for a generic component case, and later on Section 5.3 expose how they are grouped together to complete the LTI system formulation.

Most of the content and nomenclature is built-off from the basic model exposed on [3], embracing the corresponding modifications for the the introduction of airborne rerouting.

5.1.1 State Propagation for a network component

Let $\mathbf{x}_k = [{}^1x_k \; {}^2x_k \dots {}^{N_n}x_k]^T \in \mathbb{N}^{N_n \times 1}$ be the state of our system at time t_k, which represents every node's occupancy (number of aircraft dwelling on the node) at this time. Occupancy values, given its physical meaning, can only be positive integers.

Since the edges on the graph are tailored such that all of them have the same traversal time, for every timestep Δt the node occupancy on every subgraph is shifted one node forward. The state propagation matrix in every route (subgraph) i takes therefore the form of a sub-diagonal $N_i \times N_i$ square matrix, $\mathcal{A}_i$. These state propagation matrices for every route are then grouped together in a block-diagonal structure to constitute the state matrix $\mathcal{A}^{comp}$. The general case for a network component with N_R routes can be seen in 5.1.

$$\mathcal{A}^{comp} = \begin{bmatrix} \mathcal{A}_1 & 0 & 0 & \dots & 0 \\ 0 & \mathcal{A}_2 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & \mathcal{A}_{N_R} \end{bmatrix}, \quad \text{with} \quad \mathcal{A}_i = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix} \tag{5.1}$$

The network component state matrix $\mathcal{A}^{comp} \in \mathbb{N}^{N_n \times N_n}$ is a block-diagonal, constant, binary, system dynamics matrix. When an aircraft is placed on a route's node, $\mathcal{A}^{comp}$ propagates the aircraft to the subsequent node of the route each time-step. The network component state equation including only the dynamics part of states propagation from t_k to t_{k+1} can be synthesized to

$$\mathbf{x}^{comp}_{k+1} = \mathcal{A}^{comp}\mathbf{x}^{comp}_{k} \tag{5.2}$$

5.1.2 Scheduled Departures

The scheduled departures at t_{k+1} are represented by the vector $\mathbf{d}_{k+1} \in \mathbb{N}^{N_{dep} \times 1}$, where N_{dep} is the number of origin nodes (i.e. departure nodes for internal and exiting and initial nodes within the network for entering and overflights). Note that every network component has scheduled departures, even if they are not from physical airports; in the case of entering and overflights, scheduled departures represent the flights incorporation to the NAS.

The departures matrix for a network component, $\mathcal{D}^{comp} \in \mathbb{N}^{N_{dep} \times N_{dep}}$, is a constant matrix that maps the scheduled departures at t_{k+1} onto the state at t_{k+1}, i.e. $\mathcal{D}^{comp}$ places a departing aircraft at the first node of a route.

The network component state equation from t_k to t_{k+1} can be now written with the whole

uncontrolled dynamics part

$$\mathbf{x}_{k+1}^{comp} = \mathcal{A}^{comp}\mathbf{x}_{k}^{comp} + \mathcal{D}^{comp}\mathbf{d}_{k+1}^{comp} \tag{5.3}$$

5.1.3 Time-comprehensive state equation for an uncontrolled network component

Eq. 5.2 applied recursively allows propagating a system state multiple timesteps. For instance $\mathbf{x}_{k+2} = \mathcal{A}(\mathcal{A}\mathbf{x}k) = \mathcal{A}^2\mathbf{x}_k$, and with the same rationale we can express any future time t_{N_t} state given an initial state and applying the recursion N_t times.

In order to feed the data into a LP program, the time variable needs to be ruled out explicitly. With that purpose we group the recursions for every timestep on our timespan into a block-diagonal matrix to write the state propagation matrix in our LTI system with the time in explicit form.

For the Departure matrix, the procedure is similar but leading to a lower diagonal block matrix. The values of $\mathcal{A}^k D$ on the lower diagonal account for the propagation of the scheduled departures through the route after they have been placed on the first node.

For the whole timespan, the component state vectors for times $t_1, t_2, ..., t_{N_t}$ can be expressed as (Eq. 5.4)

$$\begin{bmatrix} \mathbf{x}(t_1) \\ \mathbf{x}(t_2) \\ \vdots \\ \mathbf{x}(t_{N_t}) \end{bmatrix} = \begin{bmatrix} \mathcal{A} & 0 & 0 & \dots & 0 \\ 0 & \mathcal{A}^2 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & \mathcal{A}^{N_t} \end{bmatrix} \begin{bmatrix} \mathbf{x_0} \\ \mathbf{x_0} \\ \vdots \\ \mathbf{x_0} \end{bmatrix} + \begin{bmatrix} \mathcal{D} & 0 & \dots & 0 \\ \mathcal{A}\mathcal{D} & \mathcal{D} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ \mathcal{A}^{N_t-1}\mathcal{D} & \mathcal{A}^{N_t-2}\mathcal{D} & \dots & \mathcal{D} \end{bmatrix} \begin{bmatrix} \mathbf{d}(t_1) \\ \mathbf{d}(t_2) \\ \vdots \\ \mathbf{d}(t_{N_t}) \end{bmatrix} \tag{5.4}$$

5.2 Air Traffic Flow Controls for a Network Component

The model presented in this work has available three distinct types of control. Ground holding delays the departing of an aircraft from the airport. Pre-departure rerouting and Airborne rerouting, even if seemingly being the same, possess controllability nuances that makes it preferable to keep the distinction between them. Pre-departure rerouting do not use any of the nodes retrieved by the graph extension, and it is always always bidirectional (if a reroute is feasible from 1 to 2 it would be feasible from 2 to 1).

5.2.1 Ground holding

Ground holding (gh) is a control available for the internal and exiting components. Entering and overflights don't offer the possibility of ground holding, since the airports in these cases fall outside the NAS.

${}^{gh}_{a}\mathcal{B}$ is an $N_n \times N_{dep}$ negative binary matrix. The negative ones are on the rows i affecting departure nodes ${}^{i}x$, and in the column j corresponding to the gh_j affecting it; the values elsewhere are zero. ${}^{gh}_{a}\mathcal{B}$ cancels the increment on occupancy in departure nodes, i.e. suppresses

the scheduled departures when a gh is activated.

${}^{gh}_{b}\mathcal{B}$ is an $N_n \times N_{dep}$ binary matrix with ones on the rows belonging to departure nodes in the column j corresponding to the gh_j control affecting it, and zeros elsewhere. ${}^{gh}_{b}\mathcal{B}$ places on the departure nodes the flights that have been delayed in the previous timestep.

5.2.2 Pre-departure rerouting

Pre-departure rerouting is a control available for the internal and exiting components. For analogous reasons to ground holding, it cannot affect entering nor overflights.

${}^{pr}_{a}\mathcal{B}$ is an $N_n \times N_{pr}$ matrix with values -1, 1 or 0. If ${}^{1\rightarrow4}pr$ is the $j-th$ control available, which switches a departure from node ${}^{1}x$ to take place in ${}^{4}x$, this will introduce a -1 on the first row of column j and a 1 on the fourth row of this same column. Since in our model pre-departure reroutes are bidirectional, in this case there must be also a control ${}^{4\rightarrow1}pr$ with 1 on the first row and -1 on the fourth row on its respective column. The values elsewhere are zero.

${}^{gh}_{b}\mathcal{B}$ is an $N_n \times N_{pr}$ zero matrix, since rerouting does not have an effect on future times as ground holding does.

5.2.3 Airborne rerouting

Ariborne rerouting is a control potentially availabe for all components; its availability for each component depends on the alternative routes included during the graph extensions, explained in Chapter 3 & 4.

It acts on the same fashion as pre-departure rerouting; however, as already mentioned, it is not bidirectional. ${}^{ar}_{a}\mathcal{B}$ will be an $N_n \times N_{ar}$ matrix with values -1, 1 or 0, and ${}^{ar}_{b}\mathcal{B}$ an $N_n \times N_{ar}$

zeros matrix.

5.2.4 Aggregated controls matrix and input

To create a unique control input, the different types of control available are stacked vertically. Separating the different types of controls helps to have a more structured form for $\mathcal{B}$ and is particularly handy in implementation terms when one of the controls group is set off (for instance, if we want to consider the planning problem without ground holding or any of the other two controls types).

$$\mathbf{u}(t_k) = \begin{bmatrix} \mathbf{gh}_(t_k) \\ \mathbf{pr}_(t_k) \\ \mathbf{ar}_(t_k) \end{bmatrix} \tag{5.5}$$

and the matrix introduced concatenated to create ${}_a\mathcal{B}$ and ${}_b\mathcal{B}$, both of dimension $N_n \times N_c$, which are time independent.

$$\begin{aligned} {}_b\mathcal{B} &= \begin{bmatrix} {}^{gh}_{a}\mathcal{B} & {}^{pr}_{a}\mathcal{B} & {}^{ar}_{a}\mathcal{B} \end{bmatrix} \\ {}_a\mathcal{B} &= \begin{bmatrix} {}^{gh}_{b}\mathcal{B} & \mathbf{0} & \mathbf{0} \end{bmatrix} \end{aligned} \tag{5.6}$$

For the whole timespan, the component state vectors for times $t_1, t_2, ..., t_{N_t}$ can be expressed as in Eq. 5.7

$$\begin{bmatrix} \mathbf{x}(t_1) \\ \mathbf{x}(t_2) \\ \vdots \\ \mathbf{x}(t_{N_t}) \end{bmatrix} = \mathcal{A} \begin{bmatrix} \mathbf{x_0} \\ \mathbf{x_0} \\ \vdots \\ \mathbf{x_0} \end{bmatrix} + \mathcal{D} \begin{bmatrix} \mathbf{d}(t_1) \\ \mathbf{d}(t_2) \\ \vdots \\ \mathbf{d}(t_{N_t}) \end{bmatrix} + \begin{bmatrix} {}_a\mathcal{B} & 0 & \dots & 0 \\ \mathcal{A}_a\mathcal{B} + {}_b\mathcal{B} & {}_a\mathcal{B} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ \mathcal{A}^{N_t-1}{}_a\mathcal{B} + \mathcal{A}^{N_t-2}{}_b\mathcal{B} & \mathcal{A}^{N_t-2}{}_a\mathcal{B} + \mathcal{A}^{N_t-3}{}_b\mathcal{B} & \dots & {}_a\mathcal{B} \end{bmatrix} \begin{bmatrix} \mathbf{u}(t_1) \\ \mathbf{u}(t_2) \\ \vdots \\ \mathbf{u}(t_{N_t}) \end{bmatrix} \tag{5.7}$$

5.3 Whole Network Controlled Dynamics

Finally, the different components are merged to formulate the whole network state equation. The components' vectors are stacked vertically and the matrices grouped into block-diagonal matrices

$$\begin{bmatrix} \mathbf{x}_{in} \\ \mathbf{x}_{ex} \\ \mathbf{x}_{en} \\ \mathbf{x}_{ov} \end{bmatrix} = \begin{bmatrix} \mathcal{A}_{in} & 0 & 0 & 0 \\ 0 & \mathcal{A}_{ex} & 0 & 0 \\ 0 & 0 & \mathcal{A}_{en} & 0 \\ 0 & 0 & 0 & \mathcal{A}_{ov} \end{bmatrix} \begin{bmatrix} \mathbf{x}_0^{in} \\ \mathbf{x}_0^{ex} \\ \mathbf{x}_0^{en} \\ \mathbf{x}_0^{ov} \end{bmatrix} + \begin{bmatrix} \mathcal{D}_{in} & 0 & 0 & 0 \\ 0 & \mathcal{D}_{ex} & 0 & 0 \\ 0 & 0 & \mathcal{D}_{en} & 0 \\ 0 & 0 & 0 & \mathcal{D}_{ov} \end{bmatrix} \begin{bmatrix} \mathbf{d}_{in} \\ \mathbf{d}_{ex} \\ \mathbf{d}_{en} \\ \mathbf{d}_{ov} \end{bmatrix} + \begin{bmatrix} \mathcal{B}_{in} & 0 & 0 & 0 \\ 0 & \mathcal{B}_{ex} & 0 & 0 \\ 0 & 0 & \mathcal{B}_{en} & 0 \\ 0 & 0 & 0 & \mathcal{B}_{ov} \end{bmatrix} \begin{bmatrix} \mathbf{u}_{in} \\ \mathbf{u}_{ex} \\ \mathbf{u}_{en} \\ \mathbf{u}_{ov} \end{bmatrix} \tag{5.8}$$

Equivalently, using the synthesized nomenclature, the dynamics of our network can be expressed as the following LTI system:

$$\mathbf{x} = \mathcal{A}\mathbf{X}_0 + \mathcal{D}\mathbf{d} + \mathcal{B}\mathbf{u} \tag{5.9}$$

Chapter 6

The Strategic Planning Problem

The aim of this work is to optimize the traffic flow resulting from the initial demand filed by the different airlines and airspace operators through the introduction of a group of modifications (controls) that lead to a minimization of the total delay over the network. This is referred as the Strategic Planning problem. In order to find the adequate sequence of controls to be introduced to the LTI system presented on the previous chapter, we pose the problem as an Integer Linear Programming (ILP) problem. The integer enforcement accounts for the fact that controls must be whole numbers of aircraft. The minimization is not free since the problem is subject to different constraints, both from the model consistency point of view and from physical limitations of the airspace and airport system. All of these constraints are time invariant except for the no-fly zones, mainly those arising from convective weather, which are in general time-varying as we have seen.

6.1 Constraints

The constraints reflecting the physical limitations of the airspace and airport system are modeled as capacity upper limits. It is important to underscore that capacities here are not a variable feature but a fixed threshold. Variations due to contingencies are accounted for on the form of avoidance zones, which ultimately block some routes and potentially decrease the traffic of some regions, airports included.

Again, it is strongly encouraged to consult Appendix A during the reading of this section in order to develop a better intuition of the form many of these constraints take.

6.1.1 Capacity constraints

We can classify capacity constraints into two different subsets: airport characterization and airspace characterization. The former group establish the limit on the amount of flights that an airport can dispatch simultaneously; this includes a maximum number of departures, a maximum number of arrivals, and a maximum number for the combination of both. The latter group are constraints imposed to avoid excessive concentrations of aircraft in whether a region or a specific zone. In the case of no-fly zones the capacity is simply null so that no flights are allowed on those areas. The capacity upper limits on each case will be denoted by η, with the corresponding subindex indicating the capacity to which it refers.

To formulate most of the conditions we pre-multiply our state equation (Eq. 5.9) by $\mathcal{M}_x$, also referred as picker matrix, a binary matrix whose role is to sum up the occupancy of some subset of nodes, subset that will be determined by the subindex x; it could be the nodes belonging to departing routes flying from a particular airport, the nodes within an Air Sector, or any other subset of interest.

Airport capacities

For the calculation of the airport capacities we follow the method described in [11], particularly with the procedure outlined in [4]. Historical data on each of the airports is used to find, using linear regression, a maximum total capacity for the airport. The maxima for departures and landings are set by choosing the highest value registered within the historical dataset.

Let $\mathcal{M}_{ac}$, $\mathcal{M}_{dc}$, and $\mathcal{M}_{apc}$ be the picker matrices for the number of arrivals, number of departures, and total number of aircraft constraints respectively. All of them are $(N_{ap} \cdot N_t) \times (N_n \cdot N_t)$ matrices composed by N_{ap} vertically stacked $(N_n \cdot N_t) \times (N_n \cdot N_t)$ block-diagonal matrices.

The diagonal structure responds to the fact that occupancy needs to be checked independently at every time-step. For example, on the departures constraint, in every $\mathcal{M}_{dc}$ block the values are 1 for the columns corresponding to the first nodes of routes departing from airport i, different in every of the N_{ap} stacked diagonal block-matrices, and 0 elsewhere. Since capacities are time invariant, each block of the diagonal will be identical. The other cases are analogous.

Let $\boldsymbol{\eta}_{ac}$, $\boldsymbol{\eta}_{dc}$, and $\boldsymbol{\eta}_{apc}$ be the number of arrivals, number of departures, and total number of aircraft upper limit vectors respectively. For example, on the arrivals case, $\boldsymbol{\eta}_{ac}$ is the $N_{ap} \times 1$ column vector composed by the arrival capacity ${}^{\omega}\eta_{ac}$ for every different airport ω repeated N_t times (due to time invariance the values won't change) and stacked one over the other.

- Airport arrivals capacity constraint

$$\mathcal{M}_{ac}(\mathcal{A}\mathbf{X_0} + \mathcal{D}\mathbf{d} + \mathcal{B}\mathbf{u}) \leq \boldsymbol{\eta}_{ac} \tag{6.1}$$

The only components affected are internal and exiting flights. Therefore for entering and overflights $\mathcal{M}_{ac}$ values will always be 0.

- Airport departures capacity constraint

$$\mathcal{M}_{dc}(\mathcal{A}\mathbf{X_0} + \mathcal{D}\mathbf{d} + \mathcal{B}\mathbf{u}) \leq \boldsymbol{\eta}_{dc} \tag{6.2}$$

The only components affected are internal and entering flights. Therefore for exiting and overflights $\mathcal{M}_{dc}$ values will always be 0.

- Airport total capacity constraint

$$\mathcal{M}_{apc}(\mathcal{A}\mathbf{X_0} + \mathcal{D}\mathbf{d} + \mathcal{B}\mathbf{u}) \leq \boldsymbol{\eta}_{apc} \tag{6.3}$$

All the components are affected by this constraint.

Airspace capacities

In these cases, all network components are potentially affected by every one of the constraints. These constraints are issued from the need to avoid traffic agglomeration on certain zones.

Node capacity arises from the fact that, regardless of being possible for a node to host multiple aircraft thanks to vertical stratification, the vertical space is limited. Moreover, ATCSCC segments the airspace into layers by assigning different altitudes to different types of flights, narrowing even more the number of aircraft that can be allocated on the same latitude-longitude point in practice. Sector capacity is a consequence of finite infrastructures: depending of the Air Sector, there are a certain amount of control towers capable to dispatch a limited number of flights. It is an undesirable scenario to have an amount of aircraft exceeding this limit.

Those nodes inside avoidance zones have also an extra constraint to set their occupancy to 0.

- Node capacity

$$\mathcal{M}_{nc}(\mathcal{A}\mathbf{X}_0 + \mathcal{D}\mathbf{d} + \mathcal{B}\mathbf{u}) \leq \boldsymbol{\eta}_{occ} \tag{6.4}$$

Even though node capacity is a constraint applying for all nodes, the constraint needs to be imposed only to departure and airborne junctions nodes individually; it is afterwards propagated to the rest of the nodes through the Dynamics. The picker matrix $\mathcal{M}_{nc}$ is therefore an $((N_{dep} + N_{ar}) \cdot N_t) \times (N_n \cdot N_t)$ block-diagonal matrix composed by N_t diagonal blocks of $(N_{dep} + N_{ar}) \times N_n$ matrices which have, at each row, 1 for departing or converging nodes.

Node capacity is considered homogeneous for simplicity purposes along the airspace, thus $\boldsymbol{\eta}_{occ}$ is an $(N_n \cdot N_t) \times 1$ of value η_{occ} for each element on our program.

Notice that we do not need to impose a lower bound for node capacity to prevent negative occupations; if the initial layout $\mathbf{x}_0$ and the scheduled departures $\mathbf{d}$ do not introduce this kind of inconsistencies, the constraints over controls introduced in the next subsection will indirectly imply this condition.

- Air Sector capacity

Since node capacity is imposed to be non-negative, there is no need to lower bound the sector capacity.

$$\mathcal{M}_{sc}(\mathcal{A}\mathbf{X_0} + \mathcal{D}\mathbf{d} + \mathcal{B}\mathbf{u}) \leq \boldsymbol{\eta}_{sc} \tag{6.5}$$

The picker matrix $\mathcal{M}_{sc}$ is an $(N_s \cdot N_t) \times N_n$ composed by N_t identical stacked blocks of $N_s \times N_n$ matrices which have, at each row, 1 for the nodes belonging to the sector i and 0 otherwise. $\boldsymbol{\eta}_{sc}$ is an $(N_s \cdot N_t) \times 1$ with the capacity value for each of the Air Sectors in every component.

- No-fly zones

The constraint imposing null occupancy on the avoidance zones differs from all the previously stated since it is time-dependent.

$$\mathcal{M}_{nfz}(\mathcal{A}\mathbf{X_0} + \mathcal{D}\mathbf{d} + \mathcal{B}\mathbf{u}) = 0 \tag{6.6}$$

Here $\mathcal{M}_{nfz}$ is an $1 \times (N_n \cdot N_t)$ vector, composed by N_t concatenated $1 \times N_n$ vectors with 1 on the nodes inside some of the avoidance zones and 0 elsewhere, values that will change for every of these N_t smaller vectors upon the variations of the avoidance zones over time.

It can be structured as a vector, leading to a single constraint, thanks to the model consistency constraints; the fact that controls must take whole numbers, and that the number of controls affecting a node must be less or equal than the node's occupancy, provokes that the only solution for (Eq. 6.6) is ${}^{i}x = 0$ for any node picked, i.e., null occupancy for the avoidance zones.

6.1.2 Model consistency constraints

Control input positive definiteness

$$\mathbf{u} \geq \mathbf{0} \tag{6.7}$$

Along with the fact that the problem is posed as an ILP, this enforces $\mathbf{u} \in \mathbb{N}^+$.

Total controls activated

The number of controls activated on a node cannot be greater than the occupancy value of the node. As detailed in [3], the difficulty here belies on the fact that previous controls may increase the number of aircraft available on a node at the current time. In [3] a constraint directly affecting the number of controls introduced is presented. An equivalent indirect approach is proposed here: we introduce equality constraints, similar to the first Kirchhoff law, that refer to the conservation of total number of aircraft in departing and junction nodes. This indirectly limits the introduction of controls into nodes where there are not enough aircraft available; we can introduce less controls than the number of available flights, but never more.

These controls are slightly more complex than any of the already introduced since they affect two time-steps. For example, in a diverging junction, the occupancy of the diverging node at t_k must be equal to the sum of occupancies of its children nodes at t_{k+1}.

- Ground controls: gh and pr vs. scheduled departures

gh introduced at t_k multiplies by -1 at t_k and by 1 at t_{k+1}; *pr* redirecting a flight from the node to another node, introduced at t_k, multiplies by -1 at t_k

$$ {}_g\mathcal{M}_c\mathbf{u} - \mathbf{d} = \mathbf{0} \tag{6.8} $$

where ${}_g\mathcal{M}_c$ is an $(N_{dep} \cdot N_t) \times (N_c \cdot N_t)$ with a block-diagonal and block-subdiagonal structure. The diagonal blocks are $N_{dep} \times N_c$ matrices, with a row for every departing node, and so are the sub-diagonal blocks. In the diagonal blocks there are the effects of the ground holding

and pre-departure rerouting on the nodes occupancies on the same time they are applied: if we consider one of the departing nodes under study, (one of the rows of the $N_{dep} \times N_c$ matrix constituting the diagonal block,) the element corresponding to the gh control will be -1, and any pr from this node to another one will be -1 as well; the rest of elements are 0. In the sub-diagonal blocks there are the effects of the ground holdings applied on the previous time-step. If a gh is applied on a departing node at t_k, that will increase the number of aircraft available on the same node at t_{k+1}. Therefore, for each of the rows of the $N_{dep} \times N_c$ matrix constituting the diagonal block, the values of the elements corresponding to the gh of the node will be 1 and the rest 0.

Notice that we omit flight redirections from another nodes to the node under consideration as a positive income. The intention is to avoid scenarios where multiple flights are rerouted among the same set of departing nodes leading to an unchanged balance of occupancies for every node. With a penalization for each control action in the cost function this can be also avoided, but simply ignoring incoming redirections the situation is avoided regardless of the function selected.

If there are multiple network components, they are grouped together in a block-diagonal matrix.

- Airborne controls: ar vs. original route occupancy

This condition enforces the sum of the parent nodes occupancy at t_k to match the sum of the children nodes occupancy at t_{k+1} in the ARs junctions. This will impede the activation of ar controls when there are no aircraft available.

$$_{ar}\mathcal{M}_c(\mathcal{A}\mathbf{X_0} + \mathcal{D}\mathbf{d} + \mathcal{B}\mathbf{u}) = \mathbf{0} \tag{6.9}$$

where $_{ar}\mathcal{M}_c$ is an $(N_{ar} \cdot N_t) \times (N_n \cdot N_t)$ with a block-diagonal and block-superdiagonal

structure. The superdiagonal blocks are $N_{ar} \times N_n$ matrices, with a couple of rows for every AR (one for the diverging and another for the converging junction), and so are the diagonal blocks. In the superdiagonal blocks, for every junction under consideration, there is a 1 for the children nodes. For divergent junctions there will be two of them, while for convergent only one. On the other hand, in the diagonal blocks there will be a -1 for the parent nodes. Here for divergent junctions there will be only one parent node, while for convergent two of them.

The sign rationale is arbitrary, and they could be inverted, setting the children nodes to -1 and the parents to 1. If there are multiple network components, they are grouped together in a block-diagonal matrix.

Null occupancy of ARs last node

This constraint enforces aircraft traversing an alternative route to jump back to the original route when arriving to the last AR node. As commented, first and last nodes of the ARs are "virtual" duplicated nodes which represent both the diverging and converging nodes from the original routes. This ensures every node sticks to a single route, and helps building a modular network structure, making it easier to add or remove some of the routes without affecting the rest. It is an arbitrary imposition, but it comes handy on practical implementation as the reader can see in Appendix A.

$$\mathcal{M}_{fc}(\mathcal{A}\mathbf{X_0} + \mathcal{D}\mathbf{d} + \mathcal{B}\mathbf{u}) = 0 \tag{6.10}$$

where $\mathcal{M}_{fc}$ is a $1 \times (N_n \cdot N_t)$ vector made out of N_t concatenated N_n vectors with 1 on the node corresponding to the last node of ARs and 0 elsewhere. Again, thanks to the Total Controls Activated constraints, the Null Occupancy of AR Last Nodes constraint can take

the form of a single constraint since the only solution of(Eq.6.10) is to activate the airborne reroute control for converging nodes each time an aircraft arrives to the AR last node.

If there are multiple network components, they are horizontally concatenated as well.

6.2 Cost Function

What the cost function does and form chosen (the less controls introduced the better)

$$J = \mathbf{w}^T \mathbf{u} \tag{6.11}$$

where w value for each component:

$$^i w = {}^i\delta(1 + \frac{N_t - k}{N_t})(1 + \epsilon^l) \quad \begin{cases} l = 0 & \text{if } {}^i u \text{ is } gh \\ l = 1 & \text{if } {}^i u \text{ is } pr \\ l = 2 & \text{if } {}^i u \text{ is } ar \end{cases} \tag{6.12}$$

Where ${}^i\delta$ are the number of time steps of delay introduced by control ${}^i u$, ϵ an infinitesimal positive value, and k the time step at which the control is introduced. Ground holding controls will introduce one timestep of delay, while being variable for rerouting. In the cases where the alternative route is faster, ${}^i\delta$ is set to unit to avoid negative costs. This responds not only to mathematical criteria but to the fact that in practice modifications on the scheduled plan imply logistic costs, so it is preferable to advocate for the least interventionist strategies. Following this same rationale $(1 + \epsilon^l)$ prioritizes a ground holding over a pre-departure rerouting, and a pre-departure rerouting over an airborne one. The term $(1 + \frac{N_t - k}{N_t})$ favors the introduction of controls at the end of the time horizon; since further updates for the planning information will take place, it is better to push actions for later

than to precipitate and introduce early modifications in a scenario susceptible to change.

6.3 ILP Problem

The formal definition of the Integer Linear Program is (Eq. 6.13).

$$\min_{\mathbf{u}} J = \mathbf{w}^T \mathbf{u} \quad \text{s.t} \begin{cases} \mathcal{M}_{ac}(\mathcal{A}\mathbf{X_0} + \mathcal{D}\mathbf{d} + \mathcal{B}\mathbf{u}) \leq \boldsymbol{\eta}_{ac} \\ \mathcal{M}_{dc}(\mathcal{A}\mathbf{X_0} + \mathcal{D}\mathbf{d} + \mathcal{B}\mathbf{u}) \leq \boldsymbol{\eta}_{dc} \\ \mathcal{M}_{apc}(\mathcal{A}\mathbf{X_0} + \mathcal{D}\mathbf{d} + \mathcal{B}\mathbf{u}) \leq \boldsymbol{\eta}_{apc} \\ \mathcal{M}_{nc}(\mathcal{A}\mathbf{X}_0 + \mathcal{D}\mathbf{d} + \mathcal{B}\mathbf{u}) \leq \boldsymbol{\eta}_{occ} \\ \mathcal{M}_{sc}(\mathcal{A}\mathbf{X_0} + \mathcal{D}\mathbf{d} + \mathcal{B}\mathbf{u}) \leq \boldsymbol{\eta}_{sc} \\ \mathcal{M}_{nfz}(\mathcal{A}\mathbf{X_0} + \mathcal{D}\mathbf{d} + \mathcal{B}\mathbf{u}) = 0 \\ \mathbf{u} \geq \mathbf{0} \\ {}_g\mathcal{M}_c\mathbf{u} - \mathbf{d} = \mathbf{0} \\ {}_{ar}\mathcal{M}_c(\mathcal{A}\mathbf{X_0} + \mathcal{D}\mathbf{d} + \mathcal{B}\mathbf{u}) = \mathbf{0} \\ \mathcal{M}_{fc}(\mathcal{A}\mathbf{X_0} + \mathcal{D}\mathbf{d} + \mathcal{B}\mathbf{u}) = 0 \end{cases} \tag{6.13}$$

Chapter 7

Application of the Model: Fully Worked Basic Example

7.1 Description

7.1.1 Network description

The network is constituted by five routes, with a total of thirteen nodes $\{^1x, ^2x, ..., ^{13}x\}$, and two airports, airport α and airport β. The two airports make the sole O-D pair of the network. We consider four timesteps for the planning horizon $\{t_0, t_1, ..., t_4\}$.

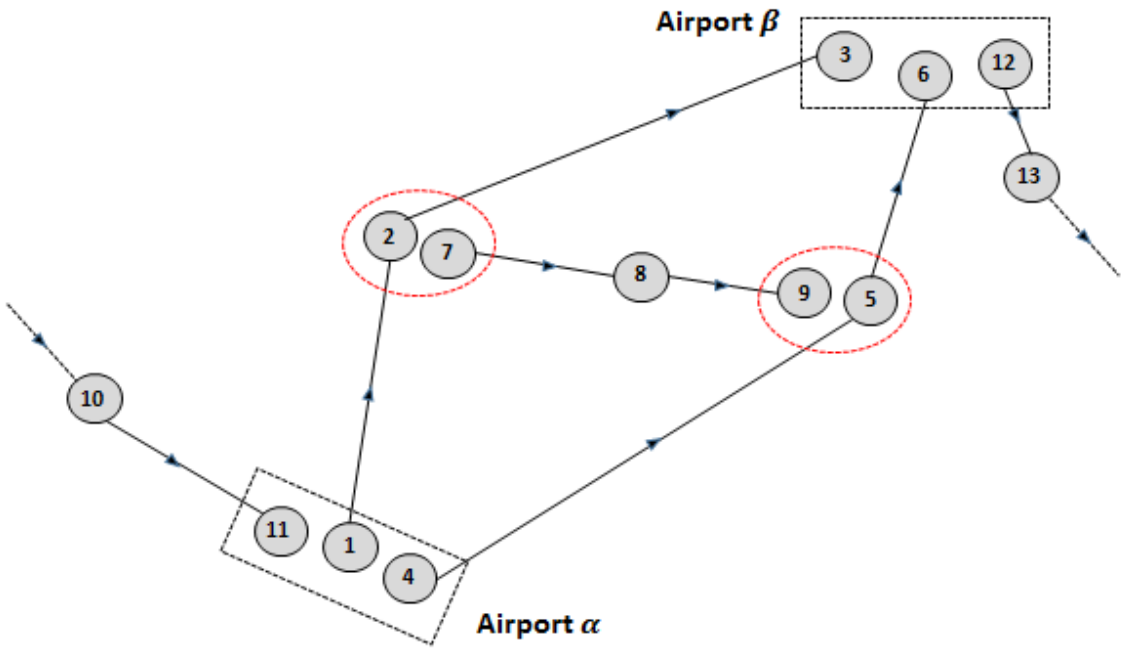


Figure 7.1: Network routes and nodes. The nodes grouped on square boxes belong to the same airport; the nodes grouped on elliptic boxes are transition pairs for airborne rerouting.

$$\begin{aligned}
\text{route } 1: &\ {}^{1}x \to {}^{2}x \to {}^{3}x \\
\text{route } 4: &\ {}^{4}x \to {}^{5}x \to {}^{6}x \\
\text{route } 1': &\ {}^{7}x \to {}^{8}x \to {}^{9}x \\
\text{route } 10: &\ {}^{10}x \to {}^{11}x \\
\text{route } 12: &\ {}^{12}x \to {}^{13}x
\end{aligned} \tag{7.1}$$

Regarding the different components of the network we distinguish:

- Three internal routes, route 1, route 4, and route 1', connecting airport α to airport β. Route 1' is an airborne reroute for route 1 to work around the blockage of ${}^{2\to3}e$ during all the time span; this reroute diverges from route 1 at node ${}^{2}x$ and converges to route 4 at node ${}^{5}x$.

- One entering route, route 10, landing at airport α.

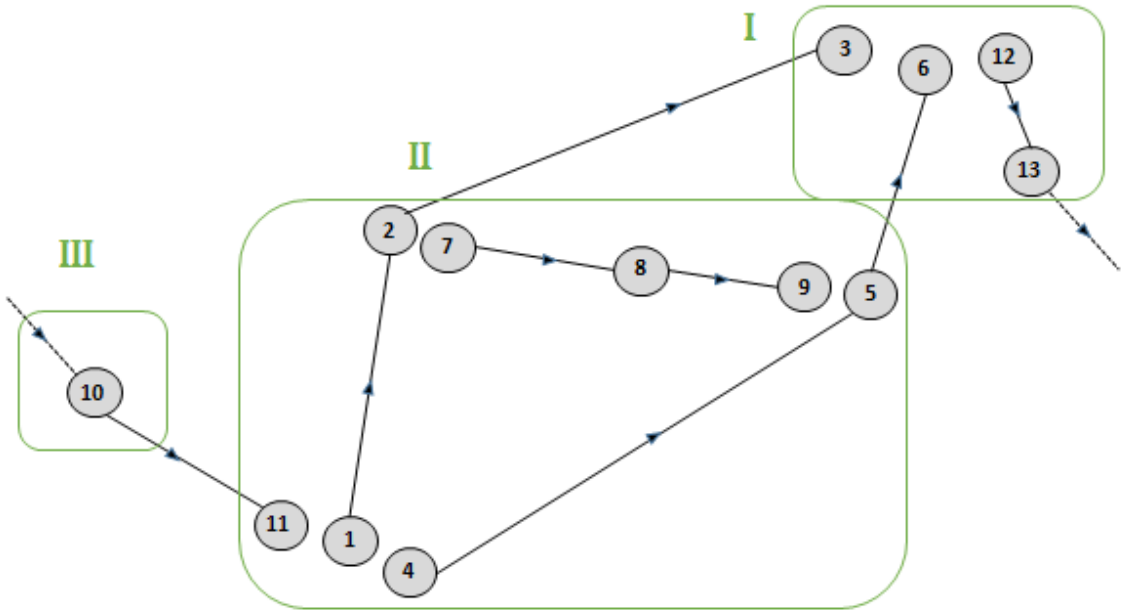


Figure 7.2: Air Sectors of the network.

- One exiting route, route 12, departing from airport β.
- No overflights.

7.1.2 Constraints description

The Air Space is divided into three sectors: the first, sector I, encompasses $\{^{3}x, ^{6}x, ^{12}x, ^{13}x\}$, the second, sector II, does the same with $\{^{1}x, ^{2}x, ^{4}x, ^{5}x, ^{7}x, ^{8}x, ^{9}x\}$, and finally the third, sector III, includes only node, ^{10}x. The capacity constraints are:

- Air Sectors I, II, and III have a capacity of 2, 3, and 1 aircraft per sector at the same time respectively.
- Airport α has a total capacity of 2 aircraft; it can launch one aircraft and land another one at the same time, but cannot launch neither land two aircraft at the same time.
- Airport β has also a total capacity of 2 aircraft, but it can land 2 flights at the same

time, launch one aircraft and land another at the same time. It cannot launch two aircraft at the same time.

The density constraints are:

- There can be up to and no more than 2 aircraft at any node on a certain time.

No-fly zones due to contingencies:

- ${}^{2\to3}e$ is blocked throughout the entire timespan due to convective weather, which sets node ${}^{3}x$ a No-fly zone for all this time.

7.1.3 Initial layout

Initial layout: at time t_0 there is one aircraft on node 1, one on node 5, and one on node 13.

7.2 Dynamics

For the sake of clarity, let's show first the dynamics for one of the network components, namely the internal flights, since the process for every component is analogous. After that we will equally state the dynamics matrices for every other of the network's components.

Remember that the state vector is also ordered by the different network components (Eq. 5.8)

for the internal flights, since the subnetwork relative to internal flights is comprised by nodes

$\{{}^1x, ...{}^9x\}$ both the state matrix and the initial state vector are:

$$\mathcal{A}^{in}_{base} = \begin{bmatrix} \mathcal{A}^1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathcal{A}^4 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathcal{A}^{1'} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}; \quad \mathbf{x}^{in}_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{7.2}$$

Regarding the matrix mapping the scheduled departing flights into routes' first nodes, $\mathcal{D}$, the only nodes that launch aircraft on the internal subnetwork are $\{{}^1x, {}^4x\}$. Vector $\mathbf{d}^{in}_i$ is the scheduled departures vector for the internal flights subnetwork at time t_i; the left superindex on its components indicates the node from which the scheduled flight will depart.

$$\mathcal{D}^{in}_{base} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}; \quad \mathbf{d}^{in}_i = \begin{bmatrix} {}^1d_{t_i} \\ {}^4d_{t_i} \end{bmatrix} \tag{7.3}$$

Considering the whole timespan

$$\mathcal{A}^{in} = \begin{bmatrix} \mathcal{A}^{in}_{base} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & (\mathcal{A}^{in}_{base})^2 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & (\mathcal{A}^{in}_{base})^3 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & (\mathcal{A}^{in}_{base})^4 \end{bmatrix}$$
$$\mathcal{D}^{in} = \begin{bmatrix} \mathcal{D}^{in}_{base} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathcal{A}^{in}_{base}\mathcal{D}^{in}_{base} & \mathcal{D}^{in}_{base} & \mathbf{0} & \mathbf{0} \\ (\mathcal{A}^{in}_{base})^2\mathcal{D}^{in}_{base} & \mathcal{A}^{in}_{base}\mathcal{D}^{in}_{base} & \mathcal{D}^{in}_{base} & \mathbf{0} \\ (\mathcal{A}^{in}_{base})^3\mathcal{D}^{in}_{base} & (\mathcal{A}^{in}_{base})^2\mathcal{D}^{in}_{base} & \mathcal{A}^{in}_{base}\mathcal{D}^{in}_{base} & \mathcal{D}^{in}_{base} \end{bmatrix} \tag{7.4}$$

The uncontrolled dynamics for the internal routes subnetwork for the entire timespan are thus:

$$\mathbf{x^{in}} = \mathcal{A}^{in} \begin{bmatrix} \mathbf{x}^{in}_0 \\ \mathbf{x}^{in}_0 \\ \mathbf{x}^{in}_0 \\ \mathbf{x}^{in}_0 \end{bmatrix} + \mathcal{D}^{in} \begin{bmatrix} \mathbf{d}^{in}_1 \\ \mathbf{d}^{in}_2 \\ \mathbf{d}^{in}_3 \\ \mathbf{d}^{in}_4 \end{bmatrix} \tag{7.5}$$

Similarly, the rest of components' dynamics are:

$$\mathcal{A}^{ex}_{base} = \mathcal{A}_{12} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}; \quad \mathbf{x}^{ex}_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$
$$\mathcal{D}^{ex}_{base} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}; \quad \mathbf{d}^{ex}_i = {}^{12}d_i \tag{7.6}$$

$$\mathcal{A}^{ex} = \begin{bmatrix} \mathcal{A}^{ex}_{base} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & (\mathcal{A}^{ex}_{base})^2 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & (\mathcal{A}^{ex}_{base})^3 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & (\mathcal{A}^{ex}_{base})^4 \end{bmatrix}$$

$$\mathcal{D}^{ex} = \begin{bmatrix} \mathcal{D}^{ex}_{base} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathcal{A}^{ex}_{base}\mathcal{D}^{ex}_{base} & \mathcal{D}^{ex}_{base} & \mathbf{0} & \mathbf{0} \\ (\mathcal{A}^{ex}_{base})^2\mathcal{D}^{ex}_{base} & \mathcal{A}^{ex}_{base}\mathcal{D}^{ex}_{base} & \mathcal{D}^{ex}_{base} & \mathbf{0} \\ (\mathcal{A}^{ex}_{base})^3\mathcal{D}^{ex}_{base} & (\mathcal{A}^{ex}_{base})^2\mathcal{D}^{ex}_{base} & \mathcal{A}^{ex}_{base}\mathcal{D}^{ex}_{base} & \mathcal{D}^{ex}_{base} \end{bmatrix} \tag{7.7}$$

$$\mathcal{A}^{en}_{base} = \mathcal{A}_{10} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}; \quad \mathbf{x}^{en}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
$$\mathcal{D}^{en}_{base} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}; \quad \mathbf{d}^{en}_i = {}^{10}d_{t_i} \tag{7.8}$$

$$\mathcal{A}^{en} = \begin{bmatrix} \mathcal{A}^{en}_{base} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & (\mathcal{A}^{en}_{base})^2 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & (\mathcal{A}^{en}_{base})^3 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & (\mathcal{A}^{en}_{base})^4 \end{bmatrix}$$

$$\mathcal{D}^{en} = \begin{bmatrix} \mathcal{D}^{en}_{base} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathcal{A}^{en}_{base}\mathcal{D}^{en}_{base} & \mathcal{D}^{en}_{base} & \mathbf{0} & \mathbf{0} \\ (\mathcal{A}^{en}_{base})^2\mathcal{D}^{en}_{base} & \mathcal{A}^{en}_{base}\mathcal{D}^{en}_{base} & \mathcal{D}^{en}_{base} & \mathbf{0} \\ (\mathcal{A}^{en}_{base})^3\mathcal{D}^{en}_{base} & (\mathcal{A}^{en}_{base})^2\mathcal{D}^{en}_{base} & \mathcal{A}^{en}_{base}\mathcal{D}^{en}_{base} & \mathcal{D}^{en}_{base} \end{bmatrix} \tag{7.9}$$

$$\mathcal{A}^{ov}_{base} = \mathcal{D}^{ov}_{base} = \varnothing \tag{7.10}$$

Notice that there are no overflight routes on our case example, therefore the state matrix for

overflights is the empty set. The uncontrolled dynamics for the whole network during the time horizon considered will be:

$$\mathbf{x} = \begin{bmatrix} \mathcal{A}^{in} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathcal{A}^{ex} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathcal{A}^{en} \end{bmatrix} \begin{bmatrix} \mathbf{x}_0^{in} \\ \mathbf{x}_0^{in} \\ \mathbf{x}_0^{in} \\ \mathbf{x}_0^{in} \\ \mathbf{x}_0^{ex} \\ \mathbf{x}_0^{ex} \\ \mathbf{x}_0^{ex} \\ \mathbf{x}_0^{ex} \\ \mathbf{x}_0^{en} \\ \mathbf{x}_0^{en} \\ \mathbf{x}_0^{en} \\ \mathbf{x}_0^{en} \end{bmatrix} + \begin{bmatrix} \mathcal{D}^{in} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathcal{D}^{ex} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathcal{D}^{en} \end{bmatrix} \begin{bmatrix} \mathbf{d}_1^{in} \\ \mathbf{d}_2^{in} \\ \mathbf{d}_3^{in} \\ \mathbf{d}_4^{in} \\ \mathbf{d}_1^{ex} \\ \mathbf{d}_2^{ex} \\ \mathbf{d}_3^{ex} \\ \mathbf{d}_4^{ex} \\ \mathbf{d}_1^{en} \\ \mathbf{d}_2^{en} \\ \mathbf{d}_3^{en} \\ \mathbf{d}_4^{en} \end{bmatrix} \tag{7.11}$$

7.3 Controls

7.3.1 Controls input to optimize

For every network component the respective control vector at time t_i is:

$$\mathbf{u}_i^{in} = \begin{bmatrix} {}^{1}gh_i \\ {}^{4}gh_i \\ {}^{1\to4}pr_i \\ {}^{4\to1}pr_i \\ {}^{1'd}ar_i \\ {}^{1'c}ar_i \end{bmatrix}; \quad \mathbf{u}_i^{ex} = {}^{12}gh_i; \quad \mathbf{u}_i^{en} = \varnothing; \quad \mathbf{u}_i^{ov} = \varnothing; \tag{7.12}$$

where gh stands for ground holding, pr for predeparture rerouting, and ar for airborne rerouting. The left superindex refers to the route ID over which this control acts; on the case of ar there is also a letter d or c to indicate if it is the node diverging or converging to one of the original network nodes.

Stacking the controls vector for each time, and subsequently doing the same with the different subnetworks we obtain the total controls vector to optimize

$$\mathbf{u} = \begin{bmatrix} \mathbf{u}_1^{in} \\ \mathbf{u}_2^{in} \\ \mathbf{u}_3^{in} \\ \mathbf{u}_4^{in} \\ \mathbf{u}_1^{ex} \\ \mathbf{u}_2^{ex} \\ \mathbf{u}_3^{ex} \\ \mathbf{u}_4^{ex} \end{bmatrix} \tag{7.13}$$

7.3.2 Controls matrix

$$\mathcal{B}^{in}_{\dashv_{base}} = \begin{bmatrix} -1 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}; \quad \mathcal{B}^{in}_{\lfloor_{base}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{7.14}$$

$$\mathcal{B}^{ex}_{\dashv_{base}} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}; \quad \mathcal{B}^{ex}_{\lfloor_{base}} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{7.15}$$

$$\mathcal{B}^{en} = \mathcal{B}^{ov} = \varnothing \tag{7.16}$$

7.3.3 Network controlled dynamics

The uncontrolled dynamics for the whole network during the time horizon considered will be:

$$\mathbf{x} = \underbrace{\begin{bmatrix} \mathcal{A}^{in} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathcal{A}^{ex} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathcal{A}^{en} \end{bmatrix}}_{\mathcal{A}} \underbrace{\begin{bmatrix} \mathbf{x}_0^{in} \\ \mathbf{x}_0^{in} \\ \mathbf{x}_0^{in} \\ \mathbf{x}_0^{in} \\ \mathbf{x}_0^{ex} \\ \mathbf{x}_0^{ex} \\ \mathbf{x}_0^{ex} \\ \mathbf{x}_0^{ex} \\ \mathbf{x}_0^{en} \\ \mathbf{x}_0^{en} \\ \mathbf{x}_0^{en} \\ \mathbf{x}_0^{en} \end{bmatrix}}_{\mathbf{X_0}} + \underbrace{\begin{bmatrix} \mathcal{D}^{in} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathcal{D}^{ex} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathcal{D}^{en} \end{bmatrix}}_{\mathcal{D}} \underbrace{\begin{bmatrix} \mathbf{d}_1^{in} \\ \mathbf{d}_2^{in} \\ \mathbf{d}_3^{in} \\ \mathbf{d}_4^{in} \\ \mathbf{d}_1^{ex} \\ \mathbf{d}_2^{ex} \\ \mathbf{d}_3^{ex} \\ \mathbf{d}_4^{ex} \\ \mathbf{d}_1^{en} \\ \mathbf{d}_2^{en} \\ \mathbf{d}_3^{en} \\ \mathbf{d}_4^{en} \end{bmatrix}}_{\mathbf{d}} + \underbrace{\begin{bmatrix} \mathcal{B}^{in} & \mathbf{0} \\ \mathbf{0} & \mathcal{B}^{ex} \end{bmatrix}}_{\mathcal{B}} \underbrace{\begin{bmatrix} \mathbf{u}_1^{in} \\ \mathbf{u}_2^{in} \\ \mathbf{u}_3^{in} \\ \mathbf{u}_4^{in} \\ \mathbf{u}_1^{ex} \\ \mathbf{u}_2^{ex} \\ \mathbf{u}_3^{ex} \\ \mathbf{u}_4^{ex} \end{bmatrix}}_{\mathbf{u}} \tag{7.17}$$

$$\mathbf{x} = \mathcal{A}\mathbf{X}_0 + \mathcal{D}\mathbf{d} + \mathcal{B}\mathbf{u} \tag{7.18}$$

7.4 Constraints

7.4.1 Capacity constraints

Ariport capacity

Left superindex denotes the airport.

- Airport arrivals

$$
\begin{aligned}
{}^{\alpha}\mathcal{M}_{ac_i}^{int} &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \forall i = 1, ..., 4 \\
{}^{\beta}\mathcal{M}_{ac_i}^{int} &= \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad \forall i = 1, ..., 4
\end{aligned} \tag{7.19}
$$

$$
\begin{aligned}
{}^{\alpha}\mathcal{M}_{ac_i}^{en} &= \begin{bmatrix} 0 & 1 \end{bmatrix} \quad \forall i = 1, ..., 4 \\
{}^{\beta}\mathcal{M}_{ac_i}^{en} &= \begin{bmatrix} 0 & 0 \end{bmatrix} \quad \forall i = 1, ..., 4
\end{aligned} \tag{7.20}
$$

$$
{}^{j}\mathcal{M}_{ac}^{comp} = \begin{bmatrix} {}^{j}\mathcal{M}_{ac_1}^{comp} & {}^{j}\mathcal{M}_{ac_2}^{comp} & {}^{j}\mathcal{M}_{ac_3}^{comp} & {}^{j}\mathcal{M}_{ac_4}^{comp} \end{bmatrix} \quad \forall\, j = \alpha, \beta;\ \forall\, comp = int, en \tag{7.21}
$$

$$
\mathcal{M}_{ac} = \begin{bmatrix} {}^{\alpha}\mathcal{M}_{ac}^{int} & {}^{\alpha}\mathcal{M}_{ac}^{en} & \mathbf{0} \\ {}^{\beta}\mathcal{M}_{ac}^{int} & {}^{\beta}\mathcal{M}_{ac}^{en} & \mathbf{0} \end{bmatrix} \tag{7.22}
$$

$$
\boldsymbol{\eta}_{ac} = \begin{bmatrix} 2 \\ 2 \\ 2 \\ 2 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{7.23}
$$

- Airport departures

$$
\begin{aligned}
{}^{\alpha}\mathcal{M}_{dc_i}^{int} &= \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \forall i = 1, ..., 4 \\
{}^{\beta}\mathcal{M}_{dc_i}^{int} &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \forall i = 1, ..., 4
\end{aligned} \tag{7.24}
$$

$$
\begin{aligned}
{}^{\alpha}\mathcal{M}^{ex}_{dc_i} &= \begin{bmatrix} 0 & 0 \end{bmatrix} \quad \forall i = 1, ..., 4 \\
{}^{\beta}\mathcal{M}^{ex}_{dc_i} &= \begin{bmatrix} 1 & 0 \end{bmatrix} \quad \forall i = 1, ..., 4
\end{aligned}
\tag{7.25}
$$

$$
{}^{j}\mathcal{M}^{comp}_{dc} = \begin{bmatrix}
{}^{j}\mathcal{M}^{comp}_{dc_1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & {}^{j}\mathcal{M}^{comp}_{dc_2} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & {}^{j}\mathcal{M}^{comp}_{dc_3} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & {}^{j}\mathcal{M}^{comp}_{dc_4}
\end{bmatrix}
\tag{7.26}
$$

$$\forall\, j = \alpha, \beta;\ \ \forall\, comp = int, ex$$

$$
\mathcal{M}_{dc} = \begin{bmatrix}
{}^{\alpha}\mathcal{M}^{int}_{dc} & \mathbf{0} & {}^{\alpha}\mathcal{M}^{ex}_{dc} \\
{}^{\beta}\mathcal{M}^{int}_{dc} & \mathbf{0} & {}^{\beta}\mathcal{M}^{ex}_{dc}
\end{bmatrix}
\tag{7.27}
$$

$$
\boldsymbol{\eta}_{dc} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}
\tag{7.28}
$$

- Airport total capacity

$$
\mathcal{M}_{apc} = \mathcal{M}_{ac} + \mathcal{M}_{dc}
\tag{7.29}
$$

$$\boldsymbol{\eta}_{apc} = \begin{bmatrix} 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \end{bmatrix} \tag{7.30}$$

Airspace capacity

- Node capacity

This constraint affects initial nodes and converging junctions from reroutes. In our case, this subset is $\{x_1, x_2, x_5, x_{10}, x_{12}\}$.

$$\begin{aligned} \mathcal{M}_{nc_i}^{int} &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \forall i = 1, ..., 4 \\ \mathcal{M}_{nc_i}^{en} &= \begin{bmatrix} 1 & 0 \end{bmatrix} \quad \forall i = 1, ..., 4 \\ \mathcal{M}_{nc_i}^{ex} &= \begin{bmatrix} 1 & 0 \end{bmatrix} \quad \forall i = 1, ..., 4 \end{aligned} \tag{7.31}$$

$$\mathcal{M}_{nc}^{comp} = \begin{bmatrix} \mathcal{M}_{nc_1}^{comp} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathcal{M}_{nc_2}^{comp} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathcal{M}_{nc_3}^{comp} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathcal{M}_{nc_4}^{comp} \end{bmatrix} \tag{7.32}$$

$$\forall\, comp = int, en, ex$$

$$\mathcal{M}_{nc} = \begin{bmatrix} \mathcal{M}_{nc}^{int} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathcal{M}_{nc}^{en} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathcal{M}_{ex}^{int} \end{bmatrix} \tag{7.33}$$

$$\boldsymbol{\eta}_{nc} = \begin{bmatrix} 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \end{bmatrix} \tag{7.34}$$

- Air Sector capacity

Left superindex denotes the Air Sector. Air Sector capacities are time-independent, therefore

$$
\begin{aligned}
{}^{I}\mathcal{M}^{int}_{ASc_i} &= \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad \forall i = 1, ..., 4 \\
{}^{II}\mathcal{M}^{int}_{ASc_i} &= \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \quad \forall i = 1, ..., 4 \\
{}^{III}\mathcal{M}^{int}_{ASc_i} &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \forall i = 1, ..., 4
\end{aligned} \tag{7.35}
$$

$$
\begin{aligned}
{}^{I}\mathcal{M}^{en}_{ASc_i} &= \begin{bmatrix} 0 & 0 \end{bmatrix} \quad \forall i = 1, ..., 4 \\
{}^{II}\mathcal{M}^{en}_{ASc_i} &= \begin{bmatrix} 0 & 1 \end{bmatrix} \quad \forall i = 1, ..., 4 \\
{}^{III}\mathcal{M}^{en}_{ASc_i} &= \begin{bmatrix} 1 & 0 \end{bmatrix} \quad \forall i = 1, ..., 4
\end{aligned} \tag{7.36}
$$

$$
\begin{aligned}
{}^{I}\mathcal{M}^{ex}_{ASc_i} &= \begin{bmatrix} 1 & 1 \end{bmatrix} \quad \forall i = 1, ..., 4 \\
{}^{I}\mathcal{M}^{ex}_{ASc_i} &= \begin{bmatrix} 0 & 0 \end{bmatrix} \quad \forall i = 1, ..., 4 \\
{}^{III}\mathcal{M}^{ex}_{ASc_i} &= \begin{bmatrix} 0 & 0 \end{bmatrix} \quad \forall i = 1, ..., 4
\end{aligned} \tag{7.37}
$$

$$
{}^{j}\mathcal{M}^{comp}_{ASc} = \begin{bmatrix} {}^{j}\mathcal{M}^{comp}_{ASc_1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & {}^{j}\mathcal{M}^{comp}_{ASc_2} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & {}^{j}\mathcal{M}^{comp}_{ASc_3} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & {}^{j}\mathcal{M}^{comp}_{ASc_4} \end{bmatrix} \tag{7.38}
$$

$$
\forall\, j = I, ..., III; \;\; \forall\, comp = int, en, ex
$$

$$
\mathcal{M}_{ASc} = \begin{bmatrix} {}^{I}\mathcal{M}^{int}_{ASc} & {}^{I}\mathcal{M}^{en}_{ASc} & {}^{I}\mathcal{M}^{ex}_{ASc} \\ {}^{II}\mathcal{M}^{int}_{ASc} & {}^{II}\mathcal{M}^{en}_{ASc} & {}^{II}\mathcal{M}^{ex}_{ASc} \\ {}^{III}\mathcal{M}^{int}_{ASc} & {}^{III}\mathcal{M}^{en}_{ASc} & {}^{III}\mathcal{M}^{ex}_{ASc} \end{bmatrix} \tag{7.39}
$$

$$\boldsymbol{\eta}_{ASc} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 2 \\ 2 \\ 2 \\ 2 \\ 3 \\ 3 \\ 3 \\ 3 \end{bmatrix} \tag{7.40}$$

- Avoidance Zones (No Fly-Zones)

$\{x_2, x_3\}$ are blocked at t_3, and $\{x_2, x_3, x_6, x_{12}\}$ at t_4. Thus

$$\begin{aligned} \mathcal{M}_{\mathcal{NFZ}_1}^{int} = \mathcal{M}_{\mathcal{NFZ}_2}^{int} &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ \mathcal{M}_{\mathcal{NFZ}_3}^{int} &= \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ \mathcal{M}_{\mathcal{NFZ}_4}^{int} &= \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \end{aligned} \tag{7.41}$$

$$\mathcal{M}_{\mathcal{NFZ}}{}^{int} = \begin{bmatrix} \mathcal{M}_{\mathcal{NFZ}_1}^{int} \, \mathcal{M}_{\mathcal{NFZ}_2}^{int} \; \mathcal{M}_{\mathcal{NFZ}_3}^{int} \; \mathcal{M}_{\mathcal{NFZ}_4}^{int} \end{bmatrix} \tag{7.42}$$

$$\begin{aligned} \mathcal{M}_{\mathcal{NFZ}_1}^{ex} = \mathcal{M}_{\mathcal{NFZ}_2}^{int} = \mathcal{M}_{\mathcal{NFZ}_3}^{ex} &= \begin{bmatrix} 0 & 0 \end{bmatrix} \\ \mathcal{M}_{\mathcal{NFZ}_4}^{ex} &= \begin{bmatrix} 1 & 0 \end{bmatrix} \end{aligned} \tag{7.43}$$

$$\mathcal{M}_{\mathcal{NFZ}}{}^{ex} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{7.44}$$

$$\mathcal{M}_{\mathcal{NFZ}_i}^{en} = \begin{bmatrix} 0 & 0 \end{bmatrix} \quad \forall i = 1, ..., 4 \tag{7.45}$$

$$\mathcal{M}_{\mathcal{NFZ}}{}^{en} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{7.46}$$

And for the whole network:

$$\mathcal{M}_{\mathcal{NFZ}} = \begin{bmatrix} \mathcal{M}_{\mathcal{NFZ}}{}^{int} \, \mathcal{M}_{\mathcal{NFZ}}{}^{ex} \;\; \mathcal{M}_{\mathcal{NFZ}}{}^{en} \end{bmatrix} \tag{7.47}$$

7.4.2 Model consistency constraints

Control input positive definiteness

$$\mathbf{u} \geq \mathbf{0} \tag{7.48}$$

Airborne rerouting consistency

- Airborne Rerouting consistency

$$\mathcal{M}_{con_i}^{in} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \forall i = 1, ..., 4 \tag{7.49}$$

$$\mathcal{M}_{con}^{int} = \begin{bmatrix} \mathcal{M}_{con_1}^{int} \, \mathcal{M}_{con_2}^{int} \;\; \mathcal{M}_{con_3}^{int} \;\; \mathcal{M}_{con_4}^{int} \end{bmatrix} \tag{7.50}$$

$$\mathcal{M}_{con_i}^{en} = \begin{bmatrix} 0 & 0 \end{bmatrix} \quad \forall i = 1, ..., 4 \tag{7.51}$$

$$\mathcal{M}_{con}^{en} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{7.52}$$

$$\mathcal{M}_{con_i}^{ex} = \begin{bmatrix} 0 & 0 \end{bmatrix} \quad \forall i = 1, ..., 4 \tag{7.53}$$

$$\mathcal{M}_{con}^{ex} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{7.54}$$

$$\mathcal{M}_{con}^{ov} = \varnothing \tag{7.55}$$

For all the entire network

$$\mathcal{M}_{con} = \begin{bmatrix} \mathcal{M}_{con}^{int} & \mathcal{M}_{con}^{ex} & \mathcal{M}_{con}^{en} \end{bmatrix} \tag{7.56}$$

Total controls activated

- Ground controls: gh and pr vs. scheduled departures.

$${}_{g}m^{ex} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \tag{7.57}$$

$${}_{g}\mathcal{M}_{c} = \begin{bmatrix} m^{int} & \mathbf{0} \\ \mathbf{0} & m^{ex} \end{bmatrix} \tag{7.58}$$

- Airborne controls: ar vs. original route occupancy.

An indirect way to ensure the number of airborne controls activated does not surpass the number of aircraft available to redirect is imposing a non-negative occupancy threshold to the ar diverging nodes. This is not necessary in general since excepting on initial nodes (solved by the other condition) and diverging junctions, propagation by dynamics preserve the occupancy along the rest of the route nodes.

$$ {}_{ar}m_i^{int} = \begin{bmatrix} 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \forall i = 2, ..., 4 \tag{7.59} $$

$$ {}_{ar}m^{int} = \begin{bmatrix} {}_{ar}m_2^{int} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & {}_{ar}m_3^{int} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & {}_{ar}m_4^{int} \end{bmatrix} \tag{7.60} $$

$$ {}_{ar}m^{ex} = \varnothing \quad {}_{ar}m^{en} = \varnothing $$

$$ {}_{ar}\mathcal{M}_c = {}_{ar}m^{int} \tag{7.61} $$

Chapter 8

Conclusions

A model for flexible redesign of route maps in contingency scenarios has been presented, concretely in the frame of an ARM. The model enhances the airspace capacity by allowing deviations from nominal routes when they are blocked due to weather events. This not only permits the ARM to recover the variety of route options lost during the aggregation step, but takes this variety of resources way further by giving it the capability to create new alternatives for scenarios not previously recorded. More importantly, this is done while preserving the low computational cost characteristic of the model, thanks to a redesign process specially tailored for route generation rather than trajectory-based.

Through an approach that is, by the best of the author's knowledge, novel in the field of Air Traffic Control, alternative routes are generated by treating the weather as a time-evolving obstacle in a sampling-based (discrete time) path planning problem, with the number of new nodes introduced to the system as the objective to minimize instead of the time delay. This last, along with the salient property of ensuring routes dynamically feasible, confers to the problem a distinction with respect to any other previous research about rerouting for Strategic Planning.

The complete controls theoretic structure for the traffic dynamics is provided, along with the integer linear program formulation to solve the optimization problem. The model is tested with real data from the NAS, showing an improvement for the total-time delay computed on the tests, almost reducing it to half of the delay for an equivalent ARM without redesign capabilities.

Bibliography

[1] Aviation Weather Center. (2017, May 21). Forecasts for Different Weather Products. Available: https://aviationweather.gov

[2] M. Ball, *et al*, "A Comprehensive Assessment of the Costs and Impacts of Flight Delay in the United States," *NEXTOR*, 2010.

[3] A. Bombelli, *et al*, "Strategic Air Traffic Planning with Fréchet distance aggregation and rerouting," *Journal of Guidance Control and Dynamics*, Vol. 40, 2017, pp. 1–12, doi: 10.2514/1.G002308.

[4] A. Bombelli, *et al*, "Automated route clustering for Air Traffic modeling," submitted for publication.

[5] T. Cormen, *et al*, *Introduction to algorithms*, 2nd ed. Cambridge, MA: MIT Press, 2001.

[6] A. Marzuoli,C. Hurter, and E. Feron, 'Data Visualization Techniques for Airspace Flow Modeling," *IEEE Conference on INtelligent Data Understanding*, Dec., 2012.

[7] D. Sun, *et al*, "Eulerian Trilogy," *AIAA Conference on Guidance, Control and Dynamics*, Aug., 2006.

[8] Eurocontrol. (2017, May 18). European airline delay cost reference values. Available: http://www.eurocontrol.int/publications/european-airline-delay-cost-reference-values

[9] Federal Aviation Administration. (2017, May 23). FAA Regulations and Policies. Available: https://www.faa.gov/regulations˙policies/orders˙notices/index.cfm/go/ document.information/documentID/73565

[10] M. Fréchet, "Sur quelques points du calcul fonctionnel," *Rendiconti del Circolo Matematico di Palermo*, Vol. 22, 1906, pp. 1–72.

[11] E. Gilbo, "Airport Capacity: Representation, Estimation, Optimization ," *IEEE Transactions on Control Systems Technology*, Vol.1, Sep., 1993, pp. 144–153.

[12] International Air Transport Association. (2017, May 28). Forecasts Passenger Demand Over Next 20 Years. Available: http://www.iata.org/pressroom/pr/Pages/2016-10-18-02.aspx

[13] R.A. Jarvis, "On the identification of the convex hull of a finite set of points in the plane," *Information Procesing Letters 2*, 1973, pp. 18–21.

[14] S. Koenig, and M. Likhachev, "D*Lite," *AAAI–02 Proceedings*, 2002, pp. 476-483.

[15] S. Koenig, and M. Likhachev, "Lifelong Planning A* and Dynamic A* Lite: The proofs," *Technical report, College of Computing, Georgia Institute of Technology, Atlanta (Georgia).*, 2001.

[16] J. Krozel, and S. Penny, "Comparison of Algorithms for Synthesizing Weather Avoidance Routes in Transition Airspace," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Aug., 2014.

[17] J. Lavalle, *Planning Algorithms*, New York, NY: Cambridge University Press, 2006.

[18] G. Law, "Quantitative Comparison of Flood Fill and Modified Flood Fill Algorithms," *International Journal of Computer Theory and Engineering*, Vol. 5, No. 3, Jun., 2014.

[19] P. Menon, *et al*, "Computer-Aided Eulerian Air Traffic Flow Modeling and Predictive Control," *Journal of Guidance Control and Dynamics*, Vol. 29, 2006, pp. 12–19.

[20] National Business Aviation Association (NBAA). (2017, May 15). Tools used for Traffic Flow Management. Available: https://www.nbaa.org/ops/airspace/tfm/tools/

[21] M. Rubnich, and R. DeLaura, "An algorithm to identify robust convective weather avoidance polygons in en route airspace," *10th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*, 2010.

[22] P. Sengupta, J. Kwan, and M. Menon, "Optimal Traffic Flow Scheduling Using High-Performance Computing," *Journal of Aerospace Information Systems*, Vol.10, 2015, pp. 661–671, doi: 10.2514/1.I010365.

[23] K. Sheth, *et al*, "Development of a Probabilistic Convective Weather Forecast Threshold Parameter for Flight-Routing Decisions," *American Meteorological Society*, Oct., 2013, pp. 1175–1187.

[24] W. Zhang, *et al*, "A hierarchical flight planning framework for air traffic management," *Proceedings of the IEEE*, Vol. 100, 2012, pp. 512–521.

[25] United States Environmental Protection Agency. (2017, May 28). Sources of Greenhouse Gas Emissions. Available: https://www.epa.gov/ghgemissions/sources-greenhouse-gas-emissions

Chapter 9

Appendix B: Sample Codes in Matlab®

For convenience, an implementation of the main algorithms discussed is attached. Please, be aware that many piping scripts are not included and these are only representative samples of the overall repository. For all the files visit `github.com/afrancir/ATMProject`.

9.1 Chapter 2 Algorithms

9.1.1 Main Avoidance Zone Generation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%$$                Arnau Franci-Rodon  8-29-2016              $$%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function allLatlonCH = ...
    PolygonsProcessorMainAsAFunction(weatherPolygons, ...
```

```
5  %     weather_latitude, weather_longitude, graphicFlag)
6  %
7  % Description: runs PolygonClassifier and PolygonsConvexHull until the
8  % polygons are stabilized and the new iterations do not alter their shape.
9  %
10 % INPUTS:
11 %         - weatherPolygons is an nxm binary matrix with values of weather
12 %           product forecast > threshold.
13 %         - graphicFlag TRUE or FALSE for graphic displaying
14 %
15 % OUTPUT:
16 %         - allLatlonCH. All M convex polygons defined by lat/lon pairs ...
       sets
17 %           for each of the N timesteps the horizon has.
18
19
20 %%% p = genpath(pwd) add this to the beginning of the Main file so every
21 %%% subfolder is accessible and ConvexPolygons can be run from the parent
22 %%% folder (and keep the files organized that way)
23
24 function allLatlonCH = ...
       PolygonsProcessorMainAsAFunctionGRID(weatherPolygons, ...
25     GRID)
26
27 nTimeSteps = size(weatherPolygons,3);
28 allLatlonCH  = cell(1,nTimeSteps);
29
30 for jj = 1:nTimeSteps
31
32     polygonsMatrix     = weatherPolygons(:,:,jj);
33     pastPolygonsMatrix = weatherPolygons.*0; %initialize it
34     nIterations        = 0;
35
```

```
36      while ¬ isequal(polygonsMatrix,pastPolygonsMatrix)
37         PolygonsClassified = PolygonClassifierEvolution2(polygonsMatrix);
38         PolygonsConvex     = PolygonsConvexHull2(PolygonsClassified);
39
40         pastPolygonsMatrix = polygonsMatrix;
41         polygonsMatrix     = PolygonsConvex;
42
43         nIterations        = nIterations + 1; %check how many does it take
44      end
45      %%% output the version with the Polygons separated from the definitive
46      %%% iteration.
47      [¬,polygonsMatrixSeparated] = PolygonsConvexHull2(PolygonsClassified);
48
49      %% Returning Polygons to lat/lon and take Convex Hull
50      latlonCH = cell(size(polygonsMatrixSeparated,3),1);
51
52      for i = 1: size(polygonsMatrixSeparated,3)
53          convex1Indices          =  find(polygonsMatrixSeparated(:,:,i)==1);
54          convex1IndicesLength    =  numel(convex1Indices);
55          weatherLatitudeConvex   =  zeros(4*convex1IndicesLength,1); %4 ...
                different lat/lon for every cell element
56          weatherLongitudeConvex =  zeros(4*convex1IndicesLength,1);
57
58          for w = 4:4:4*convex1IndicesLength
59              weatherLatitudeConvex(w-3)    = GRID{convex1Indices(w/4)}(1,2);
60              weatherLongitudeConvex(w-3)   = GRID{convex1Indices(w/4)}(1,1);
61              weatherLatitudeConvex(w-2)  = GRID{convex1Indices(w/4)}(2,2);
62              weatherLongitudeConvex(w-2) = GRID{convex1Indices(w/4)}(2,1);
63              weatherLatitudeConvex(w-1)  = GRID{convex1Indices(w/4)}(3,2);
64              weatherLongitudeConvex(w-1) = GRID{convex1Indices(w/4)}(3,1);
65              weatherLatitudeConvex(w)   = GRID{convex1Indices(w/4)}(4,2);
66              weatherLongitudeConvex(w) = GRID{convex1Indices(w/4)}(4,1);
67          end
```

```
68          K = convhull(weatherLatitudeConvex,weatherLongitudeConvex);
69          weatherLatitudeCH  = weatherLatitudeConvex(K);
70          weatherLongitudeCH = weatherLongitudeConvex(K);
71          latlonCH{i}        = [weatherLatitudeCH,weatherLongitudeCH];
72      end
73
74      allLatlonCH{1,jj} = latlonCH;
75  end %endFor
76  end %function
```

9.1.2 Closed Polygon Identifier

```
1  function PolygonsClassified = PolygonClassifierEvolution2(polygonsMatrix)
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  %$$                Arnau Franci-Rodon  8-25-2016        $$%
4  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5  % function PolygonsClassified = PolygonClassifier(polygonsMatrix)
6  % description: Closed Polygon detector, similar to flood fill algorithms.
7  %
8  % INPUT: polygonsMatrix is an nxm binary matrix
9  % OUTPUT: PolygonsClassified is an nxm integer matrix with numbers
10 % as polygons'Poly ID an NaN for non-polygonal pixels.
11 %
12 %
13 % Example:
14 %   polygonsMatrix = [1 0 1; 0 0 1; 1 1 1]
15 %   --->  PolygonsClassified = [1 0 2; 0 0 2; 2 2 2]
16
17 [row,col]          = size(polygonsMatrix);
18 PolygonsClassified = zeros(row,col);
```

```
19 classCounter       = 0;
20
21 for i = 1:row
22     for j= 1:col
23         if(polygonsMatrix(i,j)) %that is, the element is 1 and not 0
24             top = 0; left = 0;  %top, left initialization
25             % checks to avoid exceeding matrix boundaries
26             topCorner      = (i==1);
27             firstColumn    = (j==1);
28             if ¬(topCorner)
29                 top  = polygonsMatrix(i-1,j);
30             end
31             if ¬(firstColumn)
32                 left = polygonsMatrix(i,j-1);
33             end
34
35                 switcher = top + 2*left;
36
37                 switch switcher
38                     case 0 %corners were not visited yet. New polygon
39                          classCounter = classCounter + 1;
40                          PolygonsClassified(i,j) = classCounter;
41                     case 1 %only the top cell is 1, left is 0
42                         topClass  = PolygonsClassified(i-1,j);
43                         PolygonsClassified(i,j) = topClass;
44                     case 2 %left cell is 1, top is 0
45                         leftClass = PolygonsClassified(i,j-1);
46                         PolygonsClassified(i,j) = leftClass;
47                     case 3 % both are 1
48                         leftClass = PolygonsClassified(i,j-1);
49                         topClass  = PolygonsClassified(i-1,j);
50
51                         if(leftClass==topClass)
```

```
52                          PolygonsClassified(i,j) = leftClass; %or ...
                               topClass
53                      else %two polygons are in fact connected so same
54                          minClass = min([topClass,leftClass]);
55                          PolygonsClassified(i,j) = minClass;
56
57                          lS    = 0; %leftShift
58                          upS   = 0; %upShift
59
60                          while ¬isequal(topClass,leftClass,minClass)
61                              [¬,index] = ...
                                   max([topClass,leftClass,minClass]);
62                              switch index
63                                  case 1 %move up
64                                      upS = upS + 1;
65                                  case 2 %move leftwards
66                                      lS  = lS + 1;
67                                  case 3 %we reached a border
68
69                                      if any([topClass,leftClass])
70                                          display('error in the code 2')
71                                      end%if
72                                      break %break the while loop
73                              end
74                              PolygonsClassified(i-upS,j-lS) = minClass;
75                              topClass  = ...
                                   PolygonsClassified(i-(1+upS),j-lS);
76                              leftClass = ...
                                   PolygonsClassified(i-upS,j-(1+lS));
77                          end
78                      end
79                  otherwise
80                      display('error in the code')
```

```
81                  end
82          end
83      end
84  end
85  end % end function
```

9.1.3 Isolated Convex Hull Calculator

```
1  function [PolygonsConvex,PolygonsConvexApart] = ...
       PolygonsConvexHull2(polygonsClassified)
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  %$$                Arnau Franci-Rodon  8-9-2016 (m. 8-23)        $$%
4  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5  % function PolygonsConvex = PolygonsConvexHull2(polygonsClassified)
6  % description: It calculates the convex hull of every polygon labeled as
7  % independent separately, that is each polygon is isolated and then all
8  % them are overlapped back in their original position
9  %
10 % INPUT:  PolygonsClassified is an nxm integer matrix with numbers
11 % as polygons'Poly ID and 0 for non-polygonal pixels.
12 % OUTPUT: PolygonsConvex is an nxm binary matrix with the convex hull of
13 % all weather avoidance zones.
14 %
15 %
16 % Example:
17 %   PolygonsClassified = [1 0 2; 0 0 2; 2 2 2]
18 %   --->  PolygonsConvex = [1 0 0; 0 0 0; 0 0 0] + [0 0 1; 0 1 1; 1 1 1]
19 %                        = [1 0 1; 0 1 1; 1 1 1]
20
21 differentPolygonsClass = unique(polygonsClassified) ;
```

```
differentPolygonsClass = differentPolygonsClass(2:end); %0 doesn't count
nPolygons = numel(differentPolygonsClass);

%%% Then we will take the convexhull of every one of the different polygons

PolygonsApart    = zeros([size(polygonsClassified),nPolygons]);
ConvexHullsTogether = zeros([size(polygonsClassified),1]); %for ...
    graphics purposes
ConvexHullsApart = zeros([size(polygonsClassified),nPolygons]);

for k = 1:nPolygons
    PolygonsApart(:,:,k) = (polygonsClassified == ...
        differentPolygonsClass(k));
    ConvexHullsApart(:,:,k) = bwconvhull(PolygonsApart(:,:,k));
    ConvexHullsTogether = or(ConvexHullsTogether,ConvexHullsApart(:,:,k));
end

PolygonsConvex = ConvexHullsTogether; %output
PolygonsConvexApart = ConvexHullsApart; %output2

end
```

9.2 Chapter 4 Algorithms

9.2.1 Lifelong Planning A*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%$$                Arnau Franci-Rodon  8-20-2016                          $$
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
4  % function shortestPath = Main_D_Lite_asexternalfunction(nodes, edges, ...
5  %     costs, sGoal, sStart)
6  %
7  % Description: This program computes the time-dependent
8  % shortest path detour using D-Lite.
9  %
10 % INPUTS:
11 %       - nodes: nx1 vector.
12 %       - edges: mx2 matrix with all the directed connections.
13 %       - costs: mx1 vector with the corresponding costs of every.
14 %                connection.
15 %       - sGoal, sStart: scalar representing initial and final node ID.
16 % OUTPUTS:
17 %       - shortestPath: 1XN vector with the nodes' ID constituting the
18 %                       shortest path.
19 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
20
21 function shortestPath = Main_D_Lite_asexternalfunction(nodes, edges, ...
22     costs, sGoal, sStart)
23
24 Nnodes = size(nodes,1);
25 Nedges = size(edges,1);
26
27 if Nedges / size(costs,1) %throw error
28     display('Error: costs and edges dimensional mismatch')
29 end
30 % it is confortable to use Matlab's graph package, so we will be working
31 % with that
32 G = digraph(edges(:,1),edges(:,2));
33
34 %%% Procedure Initialize{}
35 U = [];
36 km = 0;
```

```
g   = Inf(Nnodes,1);
rhs = Inf(Nnodes,1);
rhs(sGoal) = 0;
U = UInsert(U,sGoal,CalculateKey(sGoal,g,rhs,sGoal,km));

shortestPath = sStart; %node ID's strain to follow for the shortest path
%%% MAIN
sLast = sStart;
% {22} Initialize()
[U,g,rhs] = ComputeShortestPath(G, U, edges, g, rhs, costs, sStart, ...
    sGoal,km);

while(sStart ~= sGoal)
    %%% {25} if g(sStart) = Inf then there is no knwon path
    sucIndex = (edges(:,1) == sStart); %successors connections
    sucIDs   = edges(sucIndex,2); %sStart successors ID

    [~,sucIDsMin]  = min(costs(sucIndex) + g(sucIDs)); %moving sStart ...
        to next cheapest
    sStart         = sucIDs(sucIDsMin);
    shortestPath = [shortestPath, sStart]; %#ok<AGROW>
    if costsChangedMarker
        km = km + h(sLast, sStart);       %#ok<UNRCH>
        sLast = sStart;
        [U,g,rhs] = ComputeShortestPath(G, U, edges, g, rhs, costs, ...
            sStart, sGoal,km);
    end
end

display(shortestPath)

end %function
```

```
function U = UInsert(U,node,key)
% function U = UInsert(U,node,key)
% description: inserts 'node' to the vertices priority queue U with ...
    priority
% established by 'key'
U = [U;[node key]];
U = sortrows(U,[2 3]);
end
```

```
function [U,rhs] = UpdateVertex(edges,U,rhs,g,costs,sGoal,u,km)
% function [U,rhs] = UpdateVertex(edges,U,rhs,g,costs,sGoal,u,km)

if u ~= sGoal
    sucIndex = (edges(:,1) == u); %successors
    sucIDs   = edges(sucIndex,2);

    rhs(u)   = min(costs(sucIndex) + g(sucIDs));
end

if any(u == U(:,1)) %if node u is in the prioirity queue
    U = URemove(U,u);
end

if g(u) ~= rhs(u)
    U = UInsert(U,u,CalculateKey(u,g,rhs,sGoal,km));
end

end
```

```
function [U,node] = UPop(U)
node = U(1,1);  % node with minimum priority

if size(U,1)==1
    U = [];
else
    U = U(2:end,:); % queue with node removed
end

end
```

```
function U = URemove(U,node)
% function U = URemove(U,node)
% description: removes node with ID #node from the priority queue U
index = find(U(:,1)==node);
U(:,1) is the list of nodes ID's present in the queue. Every node may ...
    contain as much as a
% single entrance so
if ¬isscalar(index)
    display('Error: U has multiple entrances for the same node')
end

switch index
    case 1
        U = U(2:end,:);
    case size(U,1)
        U = U(1:end-1,:);
    otherwise
        U = U([1:(index-1);(index+1):end],:);
end

```

```
20
21 end %function
```

```
1  function topNodePriority = UTop(U)
2  % function topKey = UTopKey(U)
3      if isempty(U)
4          display('UTopKey error: U is empty')
5          topNodePriority = [];
6      else
7          topNodePriority = U(1,1);
8      end
9  end
```

```
1  function topKey = UTopKey(U)
2  % function topKey = UTopKey(U)
3      if isempty(U)
4          topKey = [Inf,Inf];
5      else
6          topKey = U(1,2:3);
7      end
8  end
```

```
1  function U = UUpdate(U,node,newKey)
2  % function U = UUpdate(U,node,newKey)
3  index          = U(:,1)==node;
4  U(index,2:end) = newKey;
5  U              = sortrows(U,[2 3]);
6
7  end
```

```
function key = CalculateKey(node,g,rhs,goalNode,km)
% function key = CalculateKey(node,g,rhs,goalNode)
key = [min([g(node),rhs(node)])+ h(node,goalNode)+ km , ...
       min([g(node),rhs(node)])];
end
```

```
function [U,g,rhs] = ComputeShortestPath(G, U, edges, g, rhs, costs, ...
    sStart, sGoal,km)
% function [U,g,rhs] = ComputeShortestPath(G, U, edges, g, rhs, costs, ...
    sStart, sGoal)
%


while (isKey1GreaterKey2(CalculateKey(sStart,g,rhs,sGoal,km),UTopKey(U))...
        || rhs(sStart) ~=g(sStart) )
    kOld  = UTopKey(U);
    [U,u] = UPop(U);

    if isKey1GreaterKey2(CalculateKey(u,g,rhs,sGoal,km),kOld)
        U = UInsert(U,u,CalculateKey(u,g,rhs,sGoal,km)); ...
            %UInsert(U,node,key)
    elseif g(u)>rhs(u)
        g(u) = rhs(u);
        % for all "s" in Pred(u) --> UpdateVertex(s)
        preIDs = predecessors(G,u);
        for i = 1:numel(preIDs)
            s = preIDs(i);
            [U,rhs] = UpdateVertex(edges,U,rhs,g,costs,sGoal,s,km);
```

```
20          end
21      else
22          g(u) = Inf;
23          % for all "s" in Pred(u) & "u" itself --> UpdateVertex(s)
24          preIDsAndU = [u;predecessors(G,u)];
25          for i = 1:numel(preIDs)
26              s = preIDsAndU(i);
27              [U,rhs] = UpdateVertex(edges,U,rhs,g,costs,sGoal,s,km);
28          end
29      end
30
31  end
32
33  end %function
```

```
1  function h = h(node1, node2)
2  % function h = h(node1, node2)
3  % Description: h is an underestimate (=<) of the cost function c(node1,
4  % node2). h stands for "heuristics"
5  % For this example we take a simple norm_1:
6  h = abs(node2-node1);
7  end
```

```
1  function logical = isKey1GreaterKey2(key1,key2)
2  % Description: it compares keys in lexicographical order as explained in
3  % [14]. if key1>key2 logical=1, otherwise if key1<key2 logical=0.
4  %
5  % INPUT : key_i are 1x2 vectors.
6  % OUTPUT: logical 1 or 0.
7
```

```
8
9  if key1(1)>key2(1)
10     logical = 1;
11 elseif (key1(1)==key2(1)) && (key1(2)>key2(2))
12     logical = 1;
13 else
14     logical = 0;
15 end
16
17 end
```