FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

A Heuristic Approach to Aircraft Trajectory Optimization with Constraints

MASTERARBEIT

zur Erlangung des akademischen Grades

Master of Science

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Ing. Andreas Windbichler, BSc.
Matrikelnummer 1026414

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl

Wien, 1. Jänner 2019

Andreas Windbichler	Günther Raidl

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

A Heuristic Approach to Aircraft Trajectory Optimization with Constraints

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Computational Intelligence

by

Ing. Andreas Windbichler, BSc.
Registration Number 1026414

to the Faculty of Informatics
at the TU Wien
Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl

Vienna, 1st January, 2019

Andreas Windbichler	Günther Raidl

Erklärung zur Verfassung der Arbeit

Ing. Andreas Windbichler, BSc.
Carlbergergasse 105
1230 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. Jänner 2019

Andreas Windbichler

Kurzfassung

Der Flugverkehr nimmt kontinuierlich zu, was dazu führt, dass das Airway-Netwerk wächst und die Anzahl der Restriktionen steigt. Dadurch wird die Berechnung der effizientesten Flugbahn, einer Teilaufgabe der Flugplanung immer schwieriger. Die Fluggesellschaften versuchen unter Konkurrenzdruck ständig mehr Variablen in die Optimierungen einfließen zu lassen. Wir präsentieren deshalb eine heuristische Methode die von Beginn an die Notwendigkeit von zukünftigen Erweiterungen berücksichtigt.

Wir präsentieren eine detalierte Problemdefinition, welche, unseres Wissens nach, näher am realen Szenario liegt als frühere in der Literatur. Wir erweitern eine in der Industrie verbreitete Methode basierend auf Dijkstra's Algorithmus, welcher von uns in einen Prozess eingebettet wurde der es erlaubt komplexe Restriktionen der Luftfahrtbehörden zu berücksichtigen. Möglichst viele Restriktionen berücksichtigen wir bereits durch adaptierungen im Graphen, alle weiteren werden durch Lazy Evaluierung in einem iterativen Prozess berücksichtigt. Dieser Ansatz liefert, unter unseren Annahmen, garantiert die beste Lösung, Jedoch ist er durch den zugrundeliegenden Algorithmus nur schwer zu erweitern und die wachsende Anzahl an Restriktionen hat einen starken Einfluss auf die Laufzeit. Deshalb entwickelten wir einen heuristischen Ansatz, mit dem Ziel diese Probleme zu beseitigen. Wir machen uns die Tatsache zu nutze das die Flugbahn über den Lebensyklus eines Fluges viele male neu berechnet werden muss und somit eine Vielzahl an bereits berechneten Flugbahnen zur Verfügung steht. Diese werden von uns iterativ adaptiert um sich der neuen Situation bestmöglich anzupassen. Sofern eine Lösung eine Restriktionen verletzt, fließt dies negativ in deren Güte ein. Eine weitere Heuristik die die Gültigkeit einer gegebenen Route (unabhaengig vom Höhenprofil) bezüglich Restriktionen ausschließt, erlaubt es ganze Routen von der Suche auszuschließen.

Für unsere Tests haben wir Optimierungen zwischen europäischen Städten durchgeführt, da dieser Luftraum die größte Anzahl an Restriktionen aufweist. In diesen Tests hat der heuristische Ansatz innerhalb weniger Iterationen stets die gleichen Ergebnisse wie der exakte geliefert, dies aber in deutlich kürzerer Rechenzeit. Wir erreichten einen Speedup von bis zu 10x, und erwarten durch die steigende Anzahl von Restriktionen das dieser Abstand wachsen wird. Weiters wurde bei dem Entwurf unserer Heuristik bereits Rücksicht auf notwendige Erweiterungen genommen die im Zuge des Flightplannings notwendig sind. Diese Erweiterungen lassen sich dadurch auf natüliche Art und Weise integrieren.

Abstract

Air traffic is continuously increasing, and so is the airway network and the restrictions controlling the flow. As a result computing the most efficient trajectory, which is a subproblem of flight planning, gets harder. Airlines constantly need to consider more parameters in the optimization to stay competitive. This brings currently used algorithms to their limit. Therefore we present a heuristic method that keeps the necessary extensibility from the start in mind.

We present a detailed problem definition which comes, to our knowledge, closer to actual real-life scenario than any other in the literature. We extend an in the industry widespread approach based on Dijkstra's algorithm, which we embedded in a process to cope with Air traffic control restrictions. For all restrictions whose application can be decided based on the instance, we are adapting the graph s.t. those restrictions are already respected. All other restrictions are evaluated lazily and subsequently avoided, resulting in an iterated process. For our assumptions, this method guarantees to return the optimum result. However, because of the underlying algorithm, it is hard to extend, and the number of restrictions has a significant influence on the runtime. Therefore, we propose a heuristic method, with the goal to overcome those shortcomings. We make use of the fact that during the lifetime of a flight, the trajectory needs to be calculated several times. We use those trajectories and iteratively adapt them, to fit the changed situation best. If a solution violates a restriction, it will reflect negatively in its quality. An additional heuristic allows to preliminary determine that for a given path exists an altitude profile that does not violate any restrictions, which allows us to exclude this path from the search.

We performed tests for city pairs within Europa, which is tightly packed with restrictions. In our tests, the heuristic approach reached within a few iterations the same result as the optimal method, but within significantly less computation time. We reached speedups up to 10x and expect the margin to grow further with a growing number of restrictions. Additionally, we already considered necessary extensions in the choice of our algorithm which are necessary during the course of flight planning.

Contents

CHAPTER 1

Introduction

Air traffic is continuously increasing, and so is the stress on the airway system, see Figure 1.1 [1]. As a result, the airway network gets more complicated, the degree of freedom for planning flights increases, and finding the most efficient routes gets harder. Finding those is not just beneficial to airlines because it allows them to reduce costs, this also has an environmental impact since it reduces aircraft emissions [2].

For calculating a flight path there is much information and many steps required. First and foremost one needs to know the departure and destination airport. To fly from one airport to another, it is not simply allowed to take off and take whatever route one wants to take. Like roads spanning the ground of the earth, there is a given network in the sky as well.

This network is managed by Air Traffic Control (ATC) which is publishing waypoints that are connected by segments. Going from one waypoint to another is only allowed if both waypoints share a common segment. To manage the flow of the traffic, ATC is also putting limitations on the usability of points and segments. This means that certain points or segments are only allowed to use if the route of the aircraft fulfills specific criteria. Simple examples would be that a segment is only usable if the aircraft is departing from a particular airport, or a point is only allowed to be used if another point was not used. Without going into more detail here, there are many rules that limit the use of the airway system which will be explained later.

A path of waypoints going from the departure to the destination airport is known as a *lateral route*. In Figure 1.2 we can see the lateral route from the British Airways flight with number BA706 from the 30. November 2018 from London to Vienna.

To avoid collisions, we also have to specify an altitude profile that will be flown by the aircraft, which is called a *vertical profile*. Figure 1.3 shows the vertical profile performed by an Airbus A320-232 (corresponding to flight BA706 from Figure 1.2). This is necessary to avoid the collision of aircraft that are traversing the same segment in opposite directions

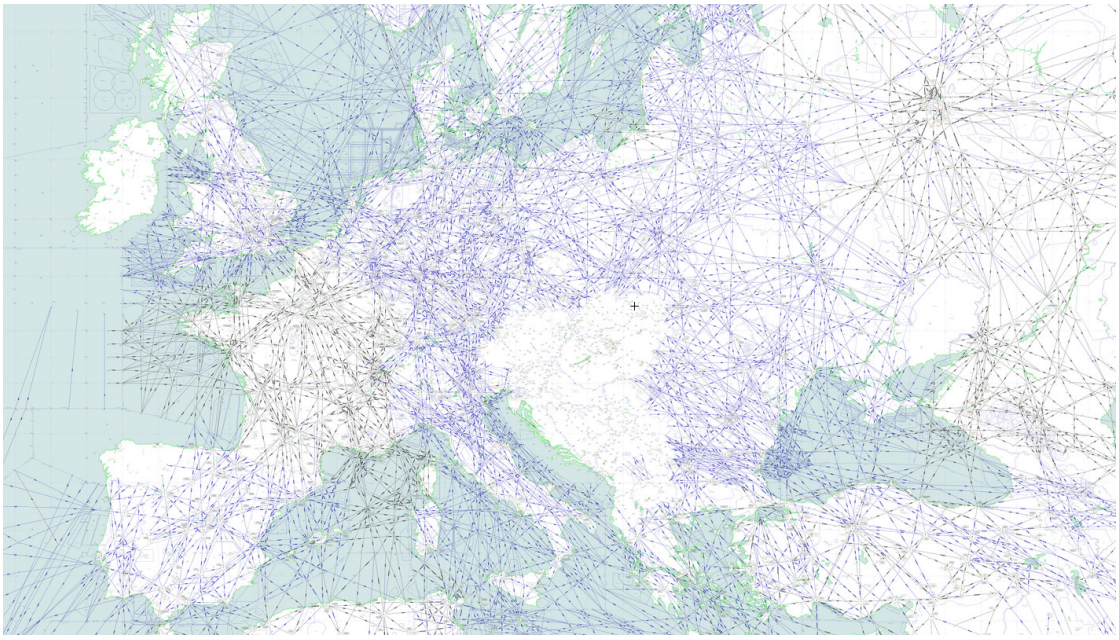


Figure 1.1: Aeronautical chart showing the European Airway system, Source SkyVector.com

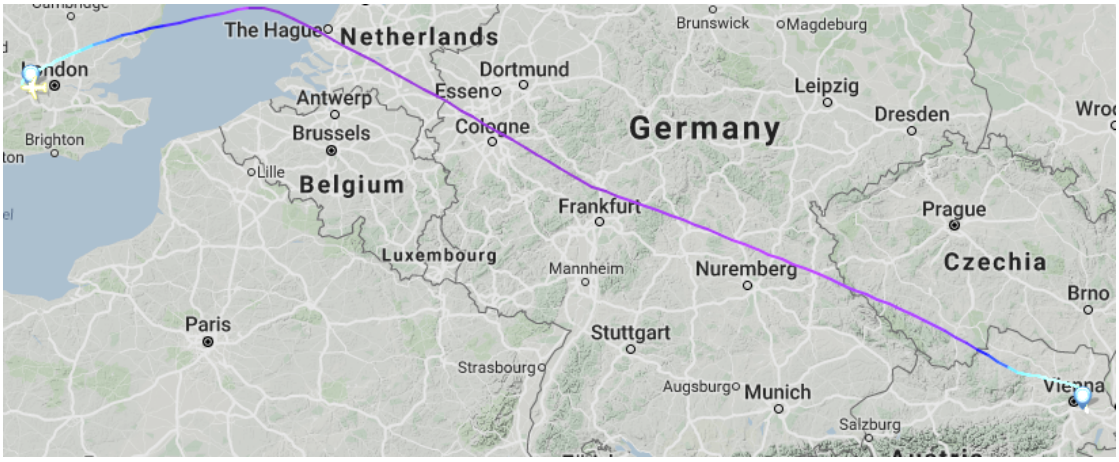


Figure 1.2: A lateral route from London to Vienna, Source: Flightradar24

as well as to allow the use of a segment by multiple aircraft at the same time in the same direction.

A lateral route combined with a vertical profile is called a *trajectory*. This trajectory, when flown by different aircraft, will result in a different fuel consumption based on the aircraft type. Not just that, it will also heavily depend on the weight the aircraft has to carry including the amount of fuel that it was tanked up with.

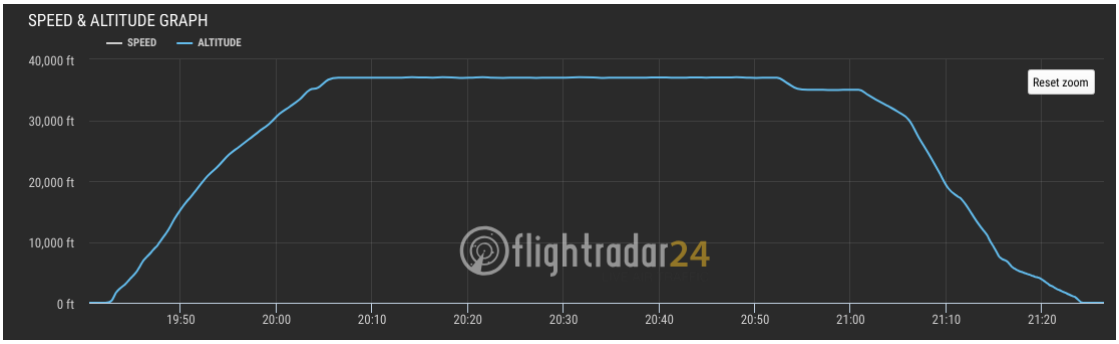


Figure 1.3: A vertical profile flown from London to Vienna, Source: Flightradar24

Another factor on the optimal trajectory is weather, most importantly the speed of the wind and the direction it is coming from. Using tailwind is beneficial for reducing the time, and fuel used to travel between points, on the other hand, headwinds will do the opposite.

Along with the weather information we also need to know the departure time. To calculate the wind along the trajectory we also need to know the time it will take us to travel a certain distance. Given the aircraft type and its initial-mass, a departure time, a weather forecast and navigation data, the goal is to find the trajectory that complies with ATC and requires the least amount of cost. To know how much fuel an aircraft will use we need a mathematical model of an aircraft.With the information about the wind and the model of the aircraft, we are finally able to calculate the total fuel consumption along the trajectory.

1.1 Motivation

In Europe, air traffic is continuously increasing, putting more and more stress on the existing airway network. This increase in traffic is followed by an increasing number of complex restrictions to the airway network from the ATC to steer the traffic in a way that it is easier for them to control. Current implementations of flight planning systems, however, are not built with such a vast amount of complex constraints to the navigation system in mind. Additionally, ATC is increasingly creating Free route airspaces (FRAs) that allow a higher degree of freedom. Since those systems are typically purely graph-based algorithms like Dijkstra's algorithm [3] or improvements like A* [4] it is hard to take advantage of this new possibility of FRA. Current systems make use of FRAs by building grids in such areas to match the input they are used to. Here they have to find a balance between a high quality in the result, by making the grid as dense as necessary, and fast optimization times by making the grid as loose as possible.

When planning a flight, one has to deal with a lot of unpredictable events. For example, weather forecasts are being used for planning, these, however, might be different compared to the actual weather when the flight is executed. The performance model used to simulate

the aircraft is an approximation and will, therefore, show differences in the real world. On top of that, the exact weight of the aircraft is not known which also influences these calculations. There can be a delay when departing which not only changes the weather along the way, but one might run into some restrictions due to the time differences arriving at each point. Therefore it is essential to have a tool that allows adapting fast to changes in the environment.

For this reason, we want to apply a genetic algorithm to a given trajectory to improve it in terms of validity and fuel consumption. The anticipated benefits of this approach are the following:

- no complete re-calculation necessary, which in contrast to Dijkstra or A* allows "hot-start"
- can be highly parallelized (by design of the algorithm)
- fast iterations with different take-off masses possible
- allows the integration of weather ensembles and show best for all, average or for specific criteria
- allows adapting to changes (like more accurate weather closer to departure and when already departed)

1.2 Related Work

Flight planning is an active field of research with enormous scope, and to our knowledge, no work considers all aspects relevant in a real-world scenario. Therefore there are many variations on what is considered in each particular work. In general, the different approaches can be categorized as follows:

- Based on optimal control theory
- Dynamic programming
- Graph-based: Dijkstra, A*

"Optimal control theory deals with the problem of finding the right inputs to a given system s.t. that an optimality criterion is reached. A control problem includes a cost functional that is a function of state and control variables. An optimal control is a set of differnetial equations describing the paths of the control variables that minimize the cost function." [5] Approaches based on optimal control theory are applied to problems where there are a short distance and much freedom. Papers using this approach usually have high accuracy in their decisions which are direct inputs to the aircraft that a pilot would make in a fighter jet or in a UAV that allows full control in the 3D space. This

method is also applied to analyze the direct avoidance of weather phenomena, again for short distances and highly accurate results. The distances of trajectories calculated by said approaches are relatively short compared to distances traveled by commercial aircraft, or they are analyzed without considering ATC restrictions. [6, 7] uses a hybrid optimal control approach to analyze the influence of wind on the optimal trajectory. Also because of the regulations and restrictions by ATC, this kind of approach does not fit for commercial aircraft in combination with a realistic real-world problem setting.

In [8] dynamic programming was used to find a vertical profile to a predefined lateral route. Similarly in [9] first a lateral path in the plane was found using Dijkstra's algorithm before applying dynamic programming to find a vertical profile. In practice, as well in the literature, it is a common approach to first find a lateral route only in the 2D space. Next, a vertical profile is generated on this lateral route, and then the speed along this trajectory is optimized. However, this does not guarantee the solution to be a global optimum for the whole 4D search space.

In [10] an approach using A* search is presented. Much research that was done for general 2D route planning, however, is based on pre-processing which is not applicable in this setting. Mendoza and Botez consider lateral navigation optimization for a fixed cruising altitude in [11]. In [12] a tunnel-like 3D grid was created around an already planned trajectory. A genetic algorithm was created to find small deviations along the initial trajectory to beneficially use wind to reduce the required fuel. Patron, Berrou and Botez improved a given trajectory by changing the vertical profile using a genetic algorithm in[13].

1.3 Contribution

Most of the existing works only tackle a restricted portion of the problem. To our knowledge there is no other work in the literature with a more extensive scope that includes the following features:

- Weather forecast model
- Aircraft performance model
- Navigation data
- ATC restrictions

All of these points are considered, while performing a full 3D optimization including an initial climb, allowing step climbs and descends, and a final descent.

Therefore the contribution of this work is to formulate a problem definition comprising the mentioned points. This problem definition is, to our knowledge, the closest to actual commercial aviation operation in the literature.

Next, an exact method that finds an optimal solution to the given problem formulation is described. Furthermore, a heuristic approach is developed in an attempt to get an algorithm that comes close the solution quality of the exact approach, while being better suited for extensions to tackle upcoming challenges (as already mentioned in this section). Finally, computational experiments are performed to compare the exact and the proposed heuristic approach.

1.4 Thesis outline

This work is structured as follows:

- Chapter 2 outlines the field of commercial aviation and gives background information necessary for flight planning.
- Chapter 3 provides a precise formulation of the flight planning problem.
- Chapter 4 a deterministic algorithm to solve the previously defined problem is presented.
- Chapter 5 contains a newly developed heuristic method for solving the problem.
- Chapter 6 presents the results of tests performed by the reference, as well as the proposed approach.
- Chapter 7 concludes the work with a discussion and future work recommendation.

CHAPTER 2

Background

2.1 Flight phases

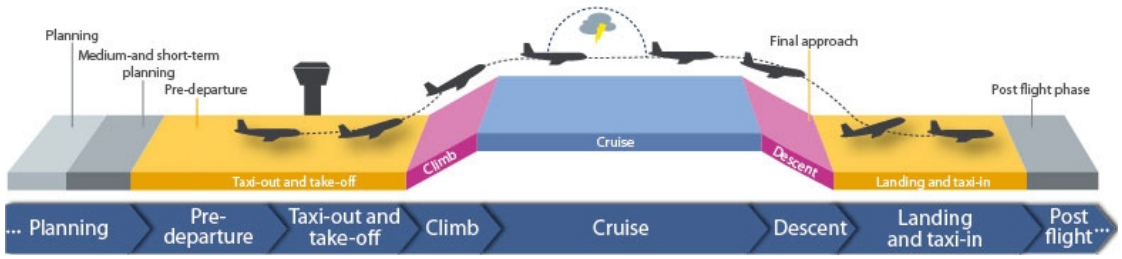


Figure 2.1: Classification of the different phases of a flight [14]

Depending on the domain of application, the phases of flight can be structured differently. Figure 2.1 shows a simplified classification from [15] which is sufficient for our use case. The first phase of flight is called taxi-out and starts at the gate, it describes the drive of the aircraft from the gate to the runway at which the aircraft is going to take off. When ready to take off the aircraft will use full thrust and start to climb. Below a certain altitude, it is typically uneconomical to fly. Therefore the pilot is going to climb as steep as possible until a more economic altitude is reached.

The point at which this altitude is reached is called Top of climb (TOC), the phase from take-off to the TOC is called *initial climb*. At this point, the passengers are allowed to remove their seatbelts and walk around in the plane. In the so-called *cruise phase* the aircraft tries to cruise on the altitude at which it requires the least cost. Over time the aircraft is losing weight because of the fuel being burned which will change the altitude at which it is most efficient to cruise. Because of this, the pilot might change the cruising altitude throughout the flight several times. Although the cruise phase is the

most efficient one, because of the longer duration in this phase it is still the one where the most fuel is burned in total numbers.

The last phase of the flight is the *final descent*, which similar to the initial climb is performed as steep as possible to reduce the duration the airplane is in a state where it consumes much fuel. The point at which the final descent is initiated is called Top of descend (TOD). From touchdown to taxing back to the gate is the last phase which we do not need to consider because this can be performed with the remaining fuel that will be there for safety purposes.

To summarize the phases of flight consists of the following:

- Taxi-out
- Initial Climb
- Cruise
- Final descent
- Landing and Taxi-in

2.2 Navigation

Official authorities, i.e. ATC, are in charge to manage regions and ensure aircrafts safety. Therefore they publish navigation data that include points (geographic locations) and connect them by so-called segments. When planning a flight, one is only allowed to travel along those published segments. This should allow the controllers to manage traffic easier. The downside is that with increasing air traffic, the stress on the airway network also increases. Looking at a single point, all aircraft aiming to fly in the region around the point are focused precisely on this location, even if it would be more cost effective not to fly directly over it.

2.2.1 Waypoints

A *waypoint* is a geographic location on the earth defined by *coordinates* (latitude and longitude). To simplify communications these waypoints are given a four-letter International Civil Aviation Organization (ICAO) code by which they can be identified. There is a whole array of different waypoint types. For this work we are only going to differentiate between airports, en route points and terminal procedure points. Airport waypoints must be at the beginning and at the end of a route.

In Table 2.1 the definition of the airport point of Vienna International Airport is shown. Compared to Table 2.2 we can see that different waypoint types have different attributes.

	Wien Schwechat (LOWW)
Coordinates:	48° 6' 37N 16° 34' 10 E
Elevation:	600ft / 183m MSL
Mag. Var.: 4.262 E	
FIR:	LOVV
ID/ ICAO:	LOWW
Type:	Civil
Runways:	2
Time Zone:	1 UTC (DST)
...	

Table 2.1: Definition of waypoint LOWW - Vienna International Airport

	Waypoint VENEN AU
Coordinates:	48° 33' 59N 14° 32' 29E
	RNAV
Mag Var:	2.235 E
FIR:	LOVV
...	

Table 2.2: Definition of waypoint VENEN

2.2.2 Segment

A *segment* is an arc connecting two points, either in only one or both directions. Note that between two points there might be multiple segments. An additional property of segments are minimum required and maximum allowed altitude. It is only allowed to traverse the segment within this altitude range.

The most important properties of a segment are:

- minimum required altitude
- maximum allowed altitude
- allowed direction: single-directional or bi-directional
- track: angle of the segment
- length: distance between its adjacent points

The track of a segment is the angle between itself and a line from the south to the north pole. In Figure 2.2 we can see a segment between PEROL and VENEN and the tracks indicated according to the direction of the segment. Because there is a discrepancy

between the true north and the magnetic north, one has to consider the magnetic variation at the waypoints in order to get the correct track.

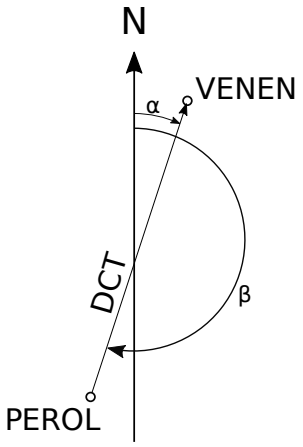


Figure 2.2: Segment PEROL → VENEN track: α, VENEN → PEROL track: β

2.2.3 Direct

A segment that connects two waypoints, but does not have any additional identifier is called a *direct*. A commonly used abbreviation for directs is DCT. In Europe, Eurocontrol is publishing every month a chart of all available DCTs in Europe (which can be found at: RAD DCT network chart). It shows all new, amended and existing DCTs.

2.2.4 Airway

An *airway* is a grouping of multiple segments using an identifier to more comfortable communication. In Table 2.3 we see the (partial) definition of the airway Z50 depicted in Figure 2.3. As we can see the airway goes from GERSA → KELIP → SOPER → PELAD → RESIA. It also defines segments in the opposite direction from RESIA → PELAD → SOPER → KELIP. Additionally, there is a non-consecutive portion AYE → AVMON → GORKA. An airway can consist of multiple non-consecutive portions that might not lie on the same continents. Its primary use is to group segments for simplified communication. If for example, someone wants to go from GERSA → KELIP → SOPER → PELAD → RESIA, one can also specify only GERSA $\xrightarrow{\text{Z50}}$ RESIA. However it is not necessary to follow an airway from beginning to end, it is also possible to only use parts of it, like KELIP $\xrightarrow{\text{X50}}$ PELAD, or even just a single segment KELIP $\xrightarrow{\text{X50}}$ SOPER.

2.2.5 Terminal procedures

As already explained there is a global network of waypoints and airways. However, those waypoints are not connected to any airports directly but, are connected via *terminal*

Sequence	Point 1	Point 2	Direction	Cruise table Identifier	Altitude minimum	Altitude maximum
10	GERSA	KELIP	→	RR	14000 ft	66000 ft
20	KELIP	SOPER	↔	RR	14000 ft	66000 ft
23	SOPER	PELAD	↔	RR	16000 ft	66000 ft
26	PELAD	RESIA	↔	RR	16000 ft	66000 ft
40	AYE	AVMON	→	RR		
50	AVMON	GORKA	→	RR		

Table 2.3: Definition of airway Z50

Figure 2.3: Aeronautical chart showing Airway Z50

procedures. To get to the network departing from an airport one has to use a Standard Instrument Departure (SID) which starts at a runway and ends at a *transition point* that is part of the global network.

As of right now, LOWW (Vienna Internation Airport) has 228 airport specific points, each of which can be used in several terminal procedures.

All the points before the transition point are airport specific and are not allowed to be used other than for the terminal procedure. Analog to the SIDs there exist procedures for getting from the global network system to an airport, which are called Standard Terminal Arrival Route (STAR)s. For easier communication, each terminal procedure has a unique ICAO code.

In Figure 2.4 we can see STAR with the ICAO code VENEN2W used for arriving at LOWW starting from VENEN which is the transition point and landing on runway RW34

in Vienna.

Table 2.4 shows more information like the specific altitude range the aircraft has to stay within. Additionally, there is much information that was omitted here for simplicity. These include a predefined speed and more detailed information of the flight path that has to be taken. Also, terminal procedures are structured into multiple parts, however, this is not of concern for this work. Additional information also specifies if a waypoint has to be flown directly over or has to be used as a reference point for a circular turn. This is essential for navigation, however the information shown here is sufficient for our use case.

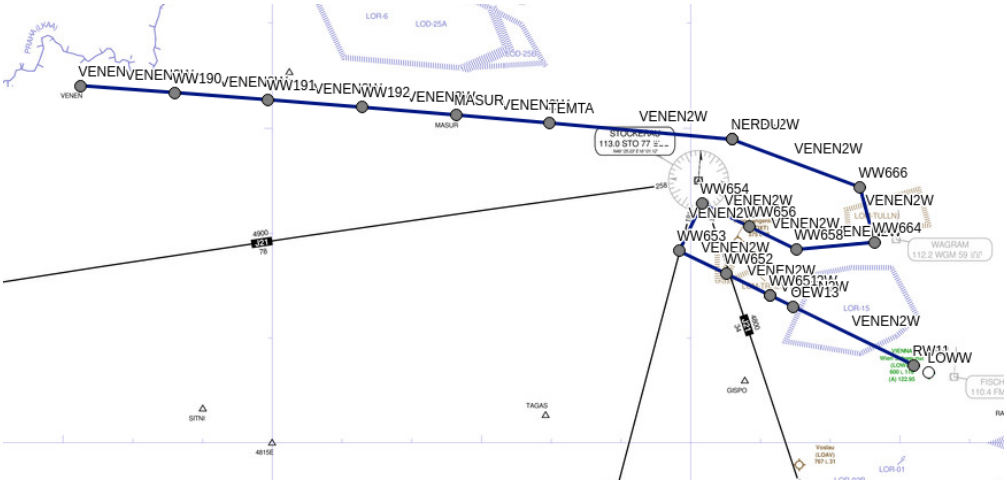


Figure 2.4: STAR VENEN2W from VENEN to LOWW Runway RW34

2.2.6 Airspace

There are different categorizations of *airspaces*, but for our purpose, it is enough to say that an airspace is a portion of the atmosphere above a country (and connected waters) which is controlled by a country.

Free route airspace - FRA

Free route airspaces are specified by a polygon that outlines the area. Along the outline, there are regular waypoints which will be referred to as boundary points. These boundary points connect it to the global airway network. In practice, there are different types of boundary points like entry, exit, and entry&exit points. The names specifies for what purposes they may be used. As a simplification in this work, all boundary points will be treated as entry&exit points, which means it is both allowed to be used for entering and exiting the FRA [16].

Sequence	Point	Altitude	
		minimum	maximum
50	MASUR	6000	17000
60	TEMTA	6000	
80	NERDU	6000	
90	NERDU	6000	
95	WW987	6000	
97	WW985	6000	
105	WW983	6000	
120	WW981	6000	
130	WW979	6000	
140	WW977	6000	
150	WW975	6000	
160	WW974	6000	
170	WW973	4000	
172	WW972	4000	
175	WW971		
180	WW971		
185	OEN74		
195	OEN40		
205	RW34		

Table 2.4: STAR VENEN2W from VENEN to LOWW Runway RW34

2.2.7 Cruise tables

When an aircraft is flying at a steady altitude, it is called cruising. To ensure safety, ATC is limiting the amount of aircraft using a particular segment at a time. Having a segment to be used by only one aircraft after the other seems wasteful. Therefore cruise tables have been introduced. A record in a *cruise table* defines at which altitudes it is allowed to cruise. The necessary information to do so is an altitude range (minimum altitude to maximum altitude) and vertical separation. For example, a minimum altitude of 1000 feet, maximum altitude 7000 feet, vertical separation 2000 ft, means that effectively it is allowed to cruise at 1000, 3000, 5000 and 7000 feet altitude.

Additionally, many segments are also available in opposite directions which introduces the risk of frontal collisions. Therefore each record of a cruise table also includes a *course from* and *course to* value which specifies for which track each record applies. Cruise tables are made up in a way that opposing traffic allowed to use alternating altitudes.

In Table 2.5 we can see that separation of 2000 feet and different “from” altitudes for courses of 0°-180° and 180°-360° are used to achieve vertical separation of 1000 feet for opposing traffic. A different visualization of the same cruise table is depicted in Figure 2.5.

As a result, ATC only has to check for possible collisions of aircraft that are changing their altitude, since aircraft are crossing cruising altitudes by doing so.

Course		True/Magnetic	Altitude		Vertical Separation
from	to		from	to	
0°	180°	M	1000 ft	41000 ft	2000 ft
0°	180°	M	45000 ft		4000 ft
180°	360°	M	2000 ft	40000 ft	2000 ft
180°	360°	M	43000 ft		4000 ft

Table 2.5: Cruise table "RR" Source A424

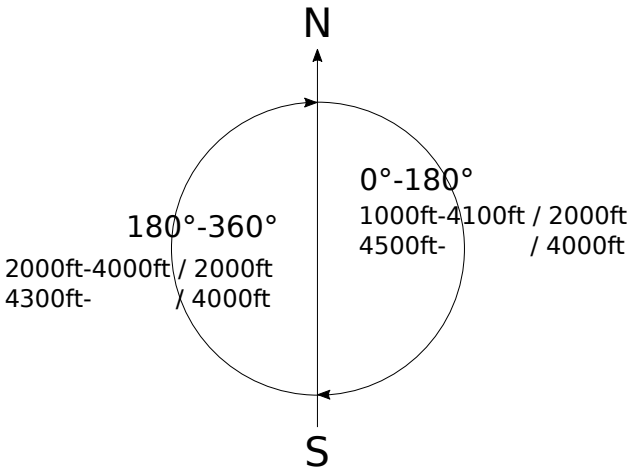


Figure 2.5: Circular visualization of the cruise table "RR" from Table 2.5

As a result of the increased air traffic, Reduced Vertical Separation Minimum (RVSM) was introduced throughout Europe which reduced the vertical separation to 1000 feet up to an altitude of 41000 feet and a separation of 2000 feet above 41000 feet. In order to use RVSM the aircraft requires specific equipment, which by now is present in almost all passenger aircraft [17].

2.2.8 Restrictions

To control the flow of aircraft, ATC uses *restrictions* to allow or forbid the use of points and segments based on certain *conditions*. ATC restrictions can be categorized into forbidding and mandatory restrictions. As the name suggests, the former tell that something is forbidden whenever a condition is true. The latter, in contrast, states that something is mandatory whenever a condition evaluates to true. Also important is that every restriction has a validity that determines at which time it is enabled. This validity refers to the actual time the aircraft is at an object (waypoint, segment, airspace, ...)

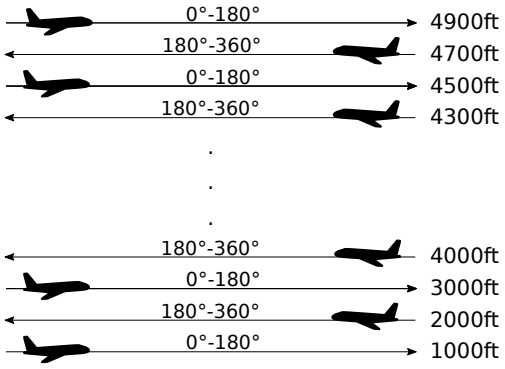


Figure 2.6: Effective cruise altitudes of the cruise table “RR”

that is restricted. In the following, we will discuss the syntax and semantics of those restrictions sufficient for the course of this thesis.

Syntax

In Table 2.6 we can see an excerpt of how such a restriction can syntactically look like. For simplicity, different predicates have been omitted in the BNF. Most notably some that checks if an aircraft is equipped with certain items or the engine type of the aircraft. A similar version including “mandatory” restrictions was defined in [18].

<restriction>	::=	<restricted-element>
		closed <opt-altitude-range> <opt-conditions>
<restricted-element>	::=	<point>
	|	
	|	<airspace>
<point>	::=	Point <point-name>
	::=	Segment <segment-name>
<airspace>	::=	Airspace <airspace-name>
<opt-altitude-range>	::=	""
	|	from <altitude> to <altitude>
<altitude>	::=	<integer> FL
<opt-conditions>	::=	""
	|	with condition <condition>
<condition>	::=	<operand>
	|	<terminal>
<operand>	::=	<or>
	|	<and>
	|	<seq>
	|	<not>
<or>	::=	or (<condition-list>)
<and>	::=	and (<condition-list>)
<seq>	::=	sequence (<condition-list>)
<not>	::=	not (<condition>)
<condition-list>	::=	<condition>
<opt-condition-list-extra>	::=	""
	|	, <condition> <opt-terminal-list-extra>
<terminal>	::=	<dep-apt>
	|	<dest-apt>
	|	<pointx>
	|	<segmentx>
	|	<airspacex>
<dep-apt>	::=	Departure_Airport <airport-icao>
<dest-apt>	::=	Destination_Airport <airport-icao>
<pointx>	::=	Point_crossing <point-name> <opt-altitude-range>
<segmentx>	::=	Segment_crossing <segment-name> <opt-altitude-range>
<airspacex>	::=	Airspace_crossing <airspace-name> <opt-altitude-range>

Table 2.6: BNF of forbid restrictions

Example 2.1 *ATC restriction according to BNF specified in Table 2.6*
Point VENEN closed
with condition or(*Departure_Airport* `LOWW`, *Point_Crossing* `MASUR`)

Semantics

In order to evaluate a restriction, we need to have a trajectory, that we can evaluate the restriction against. Figure 2.7 shows a generic trajectory (referred to as t_0 in the following section) from *Apt*1 to *Apt*2 which we will use as an example for explaining restrictions.

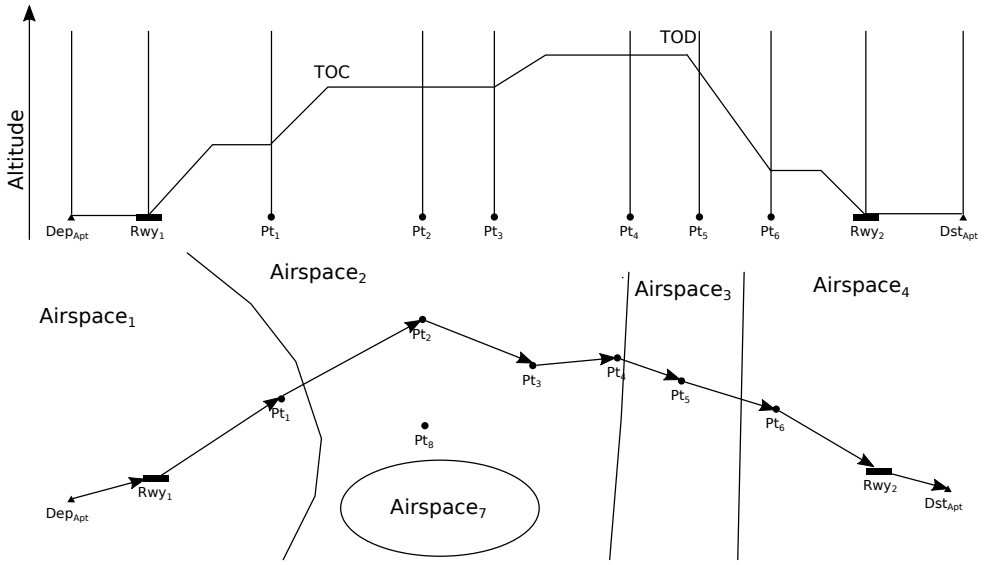


Figure 2.7: Generic trajectory for evaluation of example restrictions

A *condition* tells under which circumstance a restriction is active or inactive. In the following section the different types of conditions mentioned in 2.6 will be explained.

To this end we define the following 1-ary predicates:

$Departure\_airport_{Trajectory}$: Airport → { T, F }
allows checking whether the trajectory starts at a given airport.

$Destination\_airport_{Trajectory}$: Airport → { T, F }
allows checking whether the trajectory ends at a given airport.

$Point\_crossing_{Trajectory}$: Point → { T, F }
This condition type allows to check whether a certain point is part of the planned trajectory.

$Segment\_crossing_{Trajectory}$: Segment → { T, F }
allows checking whether the trajectory contains a given segment.

$Airspace\_crossing_{Trajectory}$: Airspace → { T, F }
allows checking whether the trajectory contains any point that is within a given airspace.

In Example 2.2, we can see the predicates for which it is not even necessary to know a trajectory to evaluate them. In contrast, the predicates from Example 2.3 require a complete trajectory to be evaluated.

Example 2.2 *For this and the follwing examples, we are again going to use the trajectory t_0 previously defined in Section 2.2.8.*

$t_0 \models Departure\_Airport\ Apt1$
$t_0 \not\models Departure\_Airport\ Apt5$
$t_0 \models Destination\_AirportApt2$
$t_0 \not\models Destination\_Airport\ Apt5$

Example 2.3 $t_0 \models Point\_crossingPt2$
$t_0 \not\models Point\_crossingPt8$
$t_0 \models Segment\_crossingDCT_{Pt1,Pt2}$
$t_0 \not\models Segment\_crossingAW1_{Pt1,Pt2}$
$t_0 \models Airspace\_crossingAirspace2$
$t_0 \not\models Airspace\_crossingAirspace7$

Logical connectives can be used to build complex condition trees. The following list is an excerpt of all the operators available. The most common ones are:

- and n-ary
- or n-ary
- not 1-ary

These and and or take an arbitrary number of truth values, while not only takes a single truth value. However, all of them are defined in the usual way, as seen in Example 2.4.

Example 2.4 *Consider again our running example of trajectory t_0 (defined in Section*

2.2.8) for the follwing evaluation.

$t_0 \models \mathsf{or}(Departure\_Airport\ Apt1, Departure\_Airport\ Apt2,$
$\qquad Departure\_Airport\ Apt3)$
$t_0 \models Departure\_Airport\ Apt1 \vee Departure\_Airport\ Apt2 \vee$
$\qquad Destination\_Airport\ Apt3$
$t_0 \models T \vee F \vee F$
$t_0 \models T$

Example 2.5 shows complete (but small) restrictions including with an explaination of their effect.

Example 2.5 *In the follwing enumeration we can see generic restrictions with a textual explaination.*

1. $Point_A$ *closed if* $\mathsf{or}(Departure\_Airport\ \texttt{LOWW}, Departure\_Airport\ \texttt{EGLL},$
   $Departure\_Airport\ \texttt{EDDF})$
   This means that the waypoint A only allowed to be used for flights which are not departing from LOWW (Vienna), EGLL (London) and EDDF (Frankfurt).

2. $DCT_{A,B}$ *closed if* $\mathsf{and}(\mathsf{not}(Departure\_Airport\texttt{LOWW}),$
   $\mathsf{not}(Destination\_Airport\ \texttt{EGLL}))$
   This means that the direct from point A to point B is only allowed to be used for flights departing from Vienna (LOWW) with destination airport London (EGLL).

3. $Point_A$ *closed if* $\mathsf{and}(\mathsf{not}(Point\ B), \mathsf{not}(Point\ C))$
   This means that the point A is only allowed to be used if the trajectory also contains point B or point C.

The last of the commonly used operators is the *sequence* operator. It is more complex than the others, because of its temporal aspect. It evaluates to true, iff there exists a partition of the trajectory such that all of its arguments evaluate to true.

$\mathsf{sequence}_{Trajectory}(Par_1, Par_2, Par_3, \ldots, Par_n) \models True$

iff $\exists t_1, t_2, t_3, \ldots, t_n : Trajectory \equiv t_1 \oplus t_2 \oplus t_3 \oplus \ldots \oplus t_n$

s.t. $t_1 \models Par_1$, $t_2 \models Par_2$, $t_3 \models Par_3$, $\ldots$, $t_n \models Par_n$

Altitude	ISA_{Temp}	$Actual_{Temp}$	Temperature Δ_{ISA}
0 m	15. °C	0 °C	-15. °C
5000 m	-17.5 °C	0 °C	+17.5 °C
10000 m	-50 °C	0 °C	+50 °C
20000 m	-115 °C	0 °C	+115 °C

Table 2.7: Example conversions from temperature in °C to Δ_{ISA}°C.

2.2.9 Publication

The full state of the navigation system is published via an Aeronautical Information Publication (AIP) by the authority of a state. The interval at which these publications appear is a 28-day cycle called the Aeronautical Information Regulation And Control (AIRAC) cycle. In between cycle dates, changes are published in the form of an amendment. The AIP already contains simple restrictions. However, the majority (and also the more complex ones) are found in the Route Availability Document (RAD) [19].

2.3 Weather

Weather plays an essential role in the execution of a flight. It could render regions unavailable for the operation because of hazardous conditions like thunderstorms or clouds of volcanic ash. Temperature changes the density of the air which influences the speed the aircraft can fly as well as the altitude it can operate. Having bad weather conditions for a route can increase the required fuel significant. On the other hand, wind can also be used to advantage to use less fuel than in still air.

2.3.1 Temperature

The International Standard Atmosphere (ISA) is an atmospheric model of how the pressure, temperature, density, and viscosity of the earth's atmosphere change over altitude. The intent for the model was to create a reference model to which the actual conditions can be put into reference. Therefore the temperature at a certain altitude can be expressed in terms of deviation to ISA conditions. Equation 2.1 shows the function for calculating the temperature in deg Celsius for an altitude in meters at ISA conditions up to an altitude of 20 kilometers (which is sufficient for commercial aircraft operation).

$$T_{ISA}(altitude) := \begin{cases} 15 - (altitude \cdot 0.0065), & 0 \leq altitude < 11km \\ -56.5, & 11km \leq altitude < 25km \end{cases} \tag{2.1}$$

Using Equation 2.1, temperature is not expressed absolute, but in terms of how much the actual temperature deviates from the temperature at ISA condition. Table 2.7 shows examples for some scenarios.

2.3.2 Wind

Since air is the medium in which the aircraft travels, the movement of the air itself has a significant influence on the effective distance the aircraft has to travel. Assuming we are flying from one point to another with a tailwind, this means that not only we are moving through the air towards the target point, but the air itself is moving as well. This means that we will be even faster at the target which means the aircraft will require less fuel than in still air. On the other hand, if you have a strong headwind, the same effect will result in longer travel time to the target point, which means more fuel required. Therefore the most fuel-efficient trajectory is heavily dependent on weather conditions.

2.3.3 Forecasts

For planning a flight in the future, it is necessary to get weather forecasts on which a calculation will be based. There are several commercial providers for weather forecasts. However, some countries meteorologic departments like the "Deutscher Wetterdienst" of Germany and government of Canda publish forecasts as a public service. The resolution goes as detailed as .25° × .25° which is more detailed than necessary for our use case. The interval in which different weather forecasts are available is 1 hours snapshots and up to 48 hours ahead of time.

2.3.4 GRIB

For distribution of weather forecasts a special format called GRIdded Binary (GRIB) was created by the World Meteorological Organization (WMO) [20]. Each GRIB file consists of multiple GRIB messages. A GRIB message contains a header that gives information about what kind of value this message contains. The values of a message itself are single values per point on a regular grid. Information on the resolution of the grid is also specified in the header. A typical GRIB could, for example, contain multiple GRIB messages per altitude. Furthermore, for every altitude it contains a message for wind speed, wind direction and ISA deviation.

2.3.5 SIGMET

Significant Meteorological Information (SIGMET) is a meteorological information service that publishes information about dangerous weather phenomena.

These weather phenomena include

- strong thunderstorms
- strong convection
- strong iceing
- strong hail

- strong turbulence
- sand and duststorm
- volcanic eruptions or volcanic ash

Regions at which such a SIGMET is published best be avoided during the planning phase of the flight.

2.4 Aircraft performance

To simulate the flight of an aircraft for a given trajectory, we need an Aircraft Performance Model (APM). In the following section, we are going to present two commonly used methods to get aircraft specific values. After that, we will introduce some general formulas for calculating the time and fuel used by the aircraft to traverse between two waypoints.

2.4.1 Tabular performance estimation

The basis of aircraft performance calculation is a so-called aircraft *performance database* typically provided by aircraft manufacturers. There is no standard for how they are formatted. Each manufacturer uses a slightly different format, all of which are in plain text format to be read by a human. An alternative way to get a performance database is to create one from actual recorded flight data. What one has to keep in mind is that this data is particular to the aircraft (performance degradation).

A performance database typically consists of at least three separate parts, which are "climb", "cruise" and "descend" [21]. Its records show a certain fuel flow for a given altitude, temperature, speed and mass. For climb and descend, it additionally contains information about the Rate of Climb (RoC) and Rate of Descent (RoD). They allow determining how steep the aircraft can climb and descend. For information on how climb and descend performance are calculated from a performance database, we want to referer to [22].

For cruise at a given altitude with a certain speed, temperature ($Temp$) and weight ($Weight$) we have to find four records in the performance database such that temperatures $Temp_1 \leq Temp \leq Temp_1$ and weights $W_1 \leq Weight \leq W_2$. Next, to get the fuel flow for a specific configuration bilinear interpolation between those records is necessary as seen in Figure 2.8. The accuracy of this approach depends on the number of records as well as the quality of the data.

2.4.2 Algorithmic performance estimation

The second approach we are going to present is to arithmetic formulas.

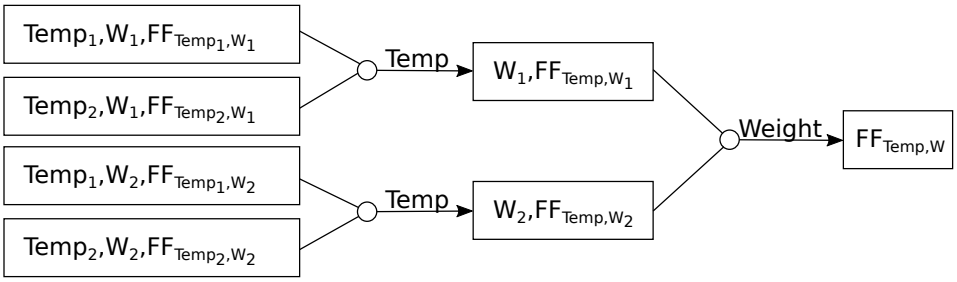


Figure 2.8: Interpolation process of performance records

Eurocontrol worked together with aircraft manufacturers to create an APM for scientific and commercial application. What they came up with is set of polynomials that approximate physical behavior of aircraft [23].

Eurocontrol has published a set of formulas that allow the approximate calculation of aircraft performance. For calculating a specific aircraft's performance, they provide a set of different coefficients for these formulas. The weight of the aircraft influences the most cost-efficient altitude for level flight. Also, the RoC and RoD are influenced by the weight of the aircraft. However, for the RoD the influence is minimal. It is necessary to consider different speeds to find the most cost-effective way to travel between two points.

2.4.3 Further calculations

To get the amount of fuel and time required to traverse between two points we have to do some further calculations. In the following section we are going to present one possible way how this can be achieved.

The True Air Speed (TAS) is the speed of an aircraft relative to the mass of air that it is traveling in. The speed of the aircraft is given as a *Mach number*, which is the speed relative to the speed of sound. Furthermore, the speed of sound, only depends on the temperature but not the pressure or density of the air. Equation 2.2 shows the fomula for speed of sound depending on the temperature (T), but all its other parameters are natural constants. Using this, the true air speed can be calculated using the Mach number and the local speed of sound (c by using the temperature at the specific location and altitude) as in Equation 2.3.

$$c_{local} = \sqrt{\frac{\gamma \cdot R \cdot T}{M}} \tag{2.2}$$

$$TAS = MachNumber \cdot c_{local} \tag{2.3}$$

The speed of the aircraft, observed by someone on the ground is called the *ground speed*. In order to calculate the ground speed, we ultimately have to take wind and its effect

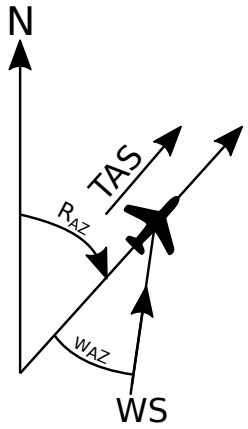


Figure 2.9: Effect of wind

(shown in Figure 2.9) into account. Equation 2.4 shows the resulting formlua, where R_{AZ} is the azimuth of the segment and W_{AZ} the azimuth of the wind. This formula is only valid as long as the speed of the wind is significantly less than the TAS, however, for this use case, the formula can safely be used.

$$GS = TAS + WS \cdot \cos(R_{AZ} - W_{AZ}) \tag{2.4}$$

Now that we know the speed relative to the ground, we have to calculate the distance that the aircraft is traveling. Since the world is not a perfectly round sphere, the World Geodetic System 1988 (WGS84) was introduced to better approximate the shape of the world. The shortest distance between two points on a sphere is called great-circle distance as shown in Figure 2.10. Algorithm 2.1 shows how to approximately compute the distance between two points on the earth using the haversine formula.

Algorithm 2.1: Distance between Points

Input: Point p_1, Point p_2
Output: Distances in km

1 $\Delta_{lat} \leftarrow \texttt{deg2rad}\,(p2_{lat} - p1_{lat})$
2 $\Delta_{lon} \leftarrow \texttt{deg2rad}\,(p2_{lon} - p1_{lon})$
3 $a \leftarrow \sin^2(\frac{\Delta_{lat}}{2}) + \cos(\texttt{deg2rad}\,(p1_{lat})\,) \cdot \cos(\texttt{deg2rad}\,(p2_{lat})\,) \cdot \sin^2(\frac{\Delta_{lon}}{2})$
4 $c \leftarrow 2 \cdot atan2(\sqrt{a}, \sqrt{1-a})$
5 $R \leftarrow 6371$ // Earth radius in km
6 **return** $6371 \cdot c$ // Distance in km

Finally, we can calculate the time (Equation 2.5) and fuel (Equation 2.6) it requires the aircraft to travel between two points. Both of these values will contribute to the total cost of the trajectory.

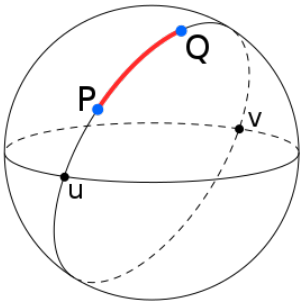


Figure 2.10: Great circle distance between P and Q [24]

$$t_{flight} = \frac{distance}{GS}; \tag{2.5}$$

$$fuelburn = fuelflow \cdot t_{flight} \tag{2.6}$$

2.5 Cost

For calculating a trajectory, several factors influence the cost of a flight. Many factors contribute to the total cost of a flight. In the following section, we are going to explain primary costs and how airlines include secondary costs into the calculation.

2.5.1 Fuel cost

The most obvious of all costs is of the fuel that is required by the aircraft to operate. It is purchased in the country where the aircraft departs. In order to keep the fuel costs as low as possible, airlines are interested in the route that requires the least amount of fuel. Also, every kilogram of fuel that goes into the airplane needs to be carried, which means that if you depart with 1 kilogram extra, does not mean that you will land with 1 kilogram more than otherwise. Therefore it is necessary to find the amount of fuel to depart with that is as low as possible while complying to all safety regulations.

2.5.2 Overflight charges

Every county charges fees in order to allow passage over its territory, which are called *overflight charges*. In general, there is no single formula for how those fees are going to be calculated, but the following covers the majority:

- Based on total distance traveled inside an airspace
- Based on distance between entry and exit point of an airspace
- Fixed one-time fee (per flight)

- Fixed fee every time the airspace is entered

2.5.3 Cost Index

As mentioned before, many secondary costs are generated by the operation of a flight. These are ranging from a simple example like the cost of the crew on board, up to cost for maintenance of the aircraft. The concept of cost index was introduced to allow taking this extra cost into account. The cost index is a measure of how expensive a minute of flight is expressed fuel (kg/min). This is not just used for calculating the cost of a route, but also by the aircraft itself. Before the flight, the pilots enter the cost-index that is going to be used in the Flight Management System (FMS) so that the aircraft can calculate the optimum speed accordingly. If for a flight a cost-index of 150 is used, this means that the for every minute during the flight, an additional amount of 150 units of fuel is treated as an extra cost. As a simple example, a Cost Index (CI) of 0 would add no additional cost, so the optimal route is the one which has the shortest Equivalent Still Air Distance (ESAD). Increasing the cost-index puts more focus on finding a faster route. Calculation of a CI within the scope, however, it needs to considered for calculating a route.

Note that some of the costs conflict because the route which requires the least amount of fuel, might not be the cheapest one regarding overflight charges or maybe takes more time increasing the cost induced by the CI.

CHAPTER 3

Problem definition

In the following chapter, we are going to define the flight planning problem we consider here as a tuple $< G, C, wf, apm, Dep_{Apt}, Dep_{Time}, Dep_{Mass}, Dst_{Apt}, CI >$ where $G = (V, A)$ is a directed multigraph, C a set of constraints on the graph, wf a weather forecast, apm an aircraft performance module, Dep_{Apt} and Dst_{Apt} departure and destination airports respectively, Dep_{Time} the departure time, Dep_{Mass} the aircraft's takeoff mass, and the cost index CI. We will use the abbreviation FPP for Flight Planning Problem.

3.0.1 Navigation

The global airway network can naturally be seen as a directed multigraph $G = (V, A)$ where the waypoints represent the nodes V and all the segments correspond to the arcs A.

Waypoint attributes:

$$coord_u \ldots \text{ coordinate of the waypoint } u \in V$$

Segment attributes:

$tail_{u,v}$ ... refers to the source u of the segment (u, v)
$head_{u,v}$ ... refers to the target v of the segment (u, v)
$alt^{min}_{u,v}$ ... minimum required altitude
$alt^{max}_{u,v}$ ... maximum allowed altitude
$alt^{crztbl}_{u,v}$ ... a set of cruise table records in the form of a tuple $(from, to, separation)$
$len_{u,v}$ ... length of the segment (determined by Algorithm 2.1)
$mid_{u,v}$ ... coordinate of the mid point

The following must hold:

$$alt_{u,v}^{min} \leq alt_{u,v}^{max} \qquad \text{for all } (u,v) \in A \tag{3.1}$$

$$\exists k : k \cdot alt_{u,v}^{sep} = alt_{u,v}^{max} - alt_{u,v}^{min} \qquad \text{for all } (u,v) \in A \tag{3.2}$$

$$mid_{u,v} := distance(coord_u, mid_{u,v}) = distance(mid_{u,v,}, coord_v)) \qquad \text{for all } (u,v) \in A \tag{3.3}$$

Where *distance* is determined by Algorithm 2.1.

3.0.2 Restrictions

Set C is the set of all forbid constraints as explained in Section 2.2.8.

3.0.3 Weather

The weather prediction function *wf* (specified in Equation 3.4) allows to query a weather forecast for a specific coordinate and altitude. The forecast contains a prediction of the wind direction, wind speed, and a temperature.

$$wf : Coordinates \times Altitudes \times Time \rightarrow Directions \times Speeds \times ISA_{\Delta} \tag{3.4}$$

3.0.4 Aircraft performance

An aircraft performance module *apm* (Equation 3.5) allows simulation of aircraft performance.

$$apm : Altitudes \times Altitudes \times Mass \times Distances \times Weather \rightarrow Altitudes \times Weight \times Durations \tag{3.5}$$

By specifying

- current altitude,
- target altitude,
- mass of the aircraft before the traversal,
- distance to travel,
- weather information (wind speed, direction, temperature)

the APM returns

- altitude reached
- fuel burned during traversal
- duration for traversal

The interface of the performance module is shown in Figure 3.1.

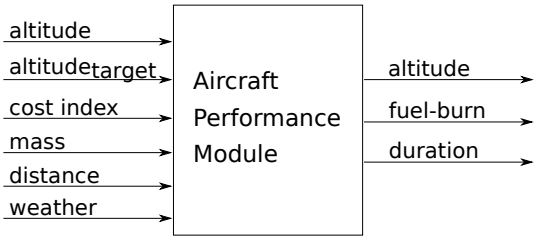
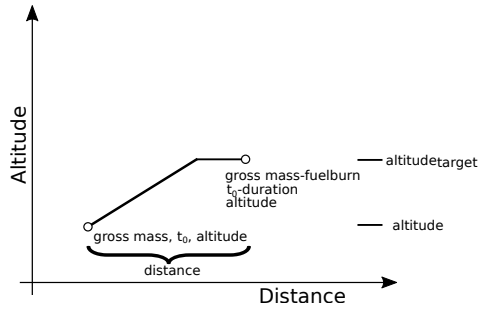
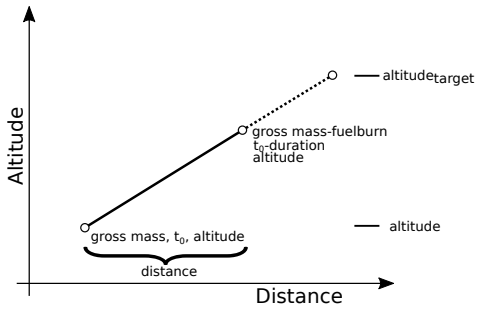


Figure 3.1: Aircraft performance module

For a climb or descend the distance might be to short for the aircraft to reach the target altitude $altitude_{target}$. In such cases, the module returns the altitude that was reached within the given distance. Another case is that the aircraft cannot perform the request, if this happens, the aircraft performance module returns an error. Reasons for the aircraft not being able to perform a request can be the altitude, mass, altitude, and combinations of them. Depending on the cost index, the optimal speed is chosen by the aircraft performance module, as it would be done by the FMS.



(a) Aircraft reaches the target altitude within the given distance



(b) The target altitude could not be reached

Figure 3.2: Example calls of the aircraft performance module and result

3.1 Trajectory

A solution to an FPP is a trajectory t in the form of Equation 3.6.

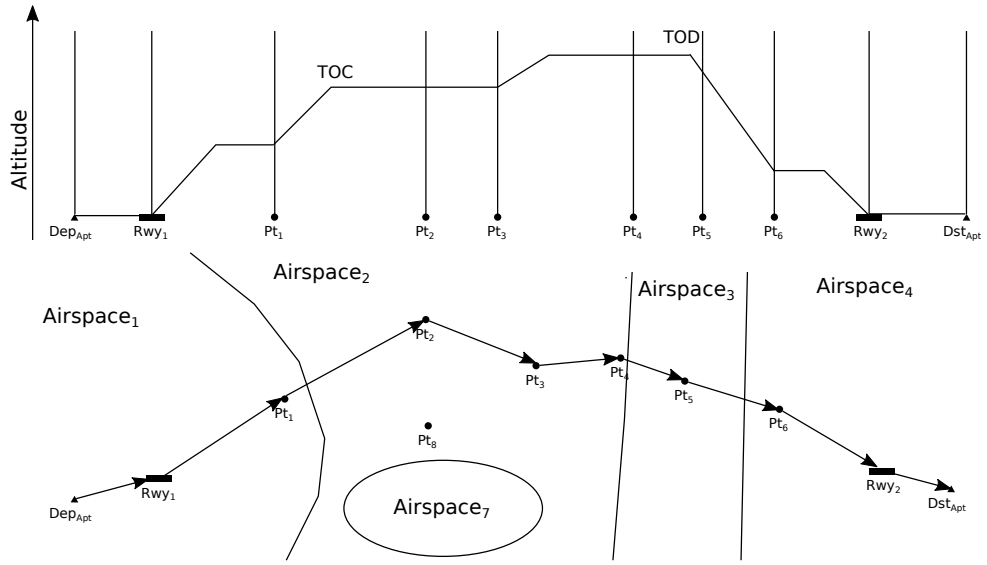


Figure 3.3: Generic trajectory for evaluation of example restrictions

$$t = [\{\underbrace{(P_1, P_2)}_{seg_1}, \underbrace{altft}_{target_1}\}, \{\underbrace{(P_2, P_3)}_{seg_2}, \underbrace{altft}_{target_2}\}, \ldots, \{\underbrace{(P_{n-1}, P_n)}_{seg_n}, \underbrace{altft}_{target_n}\}] \tag{3.6}$$

where P_1 is the departure airport and P_n the destination airport.

For a trajectory to be a valid solution to the FFP it has to respect the following constraints:

Validity w.r.t. the graph:
The trajectory describes a path from the departure airport to the destination airport, i.e. all segments (u, v) in its path:

$$(u, v) \in A \ldots \text{for all } ((u, v), target) \in Trajectory$$

Cruise tables

Cruising is only allowed on altitudes which obey the separation rule (Equation 3.7), which allows cruising on a segment (u, v) only on altitudes $\in alt^{crz}_{u,v}$.

$$alt^{crz}_{u,v} := \{alt^{min}_{u,v} + k \cdot alt^{sep}_{u,v} \mid \exists k : 0 \leq k \wedge alt^{min}_{u,v} + k \cdot alt^{sep}_{u,v} \leq alt^{max}_{u,v}\} \tag{3.7}$$

Example 3.1 *In this example we define the segment from KELIP to SOPER (and the segment in the opposite direction) including its cruise table and show on which altitudes it will be allowed to cruise at.*

- $alt^{min}_{KELIP,SOPER} = 14000ft$
- $alt^{max}_{KELIP,SOPER} = 66000ft$
- $alt^{crztbl}_{KELIP,SOPER} = \{1000ft, 41000ft, 2000ft), (45000ft, \infty ft, 4000ft)\}$
- $alt^{min}_{SOPER,KELIP} = 14000ft$
- $alt^{max}_{SOPER,KELIP} = 66000ft$
- $alt^{crztbl}_{SOPER,KELIP} = \{2000ft, 40000ft, 2000ft), (43000ft, \infty ft, 4000ft)\}$

The effectively allowed cruising altitudes are:

- $alt^{crz}_{KELIP,SOPER} = \{15000ft, 17000ft, 19000ft, \ldots, 41000ft, 45000ft, 49000ft, \ldots, 65000ft\}$
- $alt^{crz}_{SOPER,KELIP} = \{14000ft, 16000ft, 18000ft, \ldots, 40000ft, 43000ft, 47000ft, \ldots, 63000ft\}$
- $|alt^{crz}_{KELIP,SOPER}| = 20$
- $|alt^{crz}_{SOPER,KELIP}| = 20$

Example 3.1 shows the formal definition of the segment KELIP $\xrightarrow{\text{Z50}}$ SOPER (with a track of 118°) and the segment in the opposite direction SOPER $\xrightarrow{\text{Z50}}$ KELIP (with a track of 298°) from Table 2.3 (with sequence number 20) using the cruise table "RR" from Table 2.5.

Equation 3.8 enforces that a cruise can only happen on altitudes on which it is actually allowed to cruise. The aircraft does not necessarily have to cruise on the target altitude.

$$target \in alt^{crz}_{u,v} \quad \text{for all } ((u,v), target) \in Trajectory \tag{3.8}$$

Restrictions

The set C defined as part of the FPP, contains all ATC restrictions that the trajectory has to respect. For a trajectory to respect ATC restrictions no restriction must be violated. That means that for all restricted elements *elem* (waypoints, segments, airspaces) contained in the trajectory, the trajectory does not fulfill the condition c of the restriction (Equation 3.9).

$$elem \in Trajectory \implies Trajectory \not\models c \text{ for all } (elem \text{ closed if } c) \in C \tag{3.9}$$

Validity w.r.t. aircraft performance:

Let δ_n denote the state of the aircraft at waypoint P_n along a trajectory.

The initial state of the aircraft is given by the problem instance.

$$\begin{aligned}\delta^{mass}_{Dep_{Apt}} &:= Dep_{Mass}\\ \delta^{time}_{Dep_{Apt}} &:= Dep_{time}\\ \delta^{altitude}_{Dep_{Apt}} &:= Dep^{elevation}_{Apt}\end{aligned}$$

Let $\omega_{(u,v)}$ denote the weather on a segment $(u, v) \in A$ on a Trajectory given by Equation 3.10.

$$\omega(u, v) := wf(mid_{u,v}, \frac{\delta^{altitude}_u + target_v}{2}, \delta^{time}_u) \tag{3.10}$$

The previous definitions allow us to inductively define the state of the aircraft for each point of the trajectory. By utilizing the aircraft performance module and the weather forecast we can calculate, the aircrafts state for each following segment.

$$\begin{aligned}\delta_v := \delta_u \oplus apm(&\delta^{altitutde}_u, && \dots \text{altitude at the previous point}\\ &target_{altitude}, && \dots \text{target altitude for the next point}\\ &\delta^{mass}_u, && \dots \text{aircraft mass at previous point}\\ &len_{u,v}, && \dots \text{length of traversed segment}\\ &\omega(u, v)) && \dots \text{weather on the segment at the current time}\\ & && \text{for all } ((u, v), target_{altitude}) \in Trajectory\end{aligned}$$

The operation $\oplus$ is defined in Equation 3.11 in a way that the time increases by the duration the traversal takes, the mass gets reduced by the fuel burned by the aircraft and the altitude is set to the altitude reached according to the aircraft performance module.

$$\begin{aligned}\delta_{out} &:= \delta_{in} \oplus (altitude, weight, duration)\\ &\text{defined as:}\\ \delta^{altitude}_{out} &:= altitude\\ \delta^{mass}_{out} &:= \delta^{mass}_{in} - weight\\ \delta^{time}_{out} &:= \delta^{time}_{in} + duration\end{aligned} \tag{3.11}$$

The trajectory is valid w.r.t. aircraft performance if no performance query exceeds the aircraft's performance capabilities.

3.1.1 Total Costs

The total costs (Equation 3.12) of a trajectory are made up of the fuel consumed for traveling, as well as the operational cost per unit of time. Overflight charges are also a typical part of the cost of a trajectory. However, for this work we do not consider them as part of the problem.

$$totalcost := \underbrace{\delta^{mass}_{Apt_{Dst}} - \delta^{mass}_{Apt_{Dep}}}_{\text{Cost of fuel}} + \underbrace{(\delta^{time}_{Apt_{Dst}} - \delta^{time}_{Apt_{Dep}}) \cdot CI}_{\text{Cost of time}} \tag{3.12}$$

3.2 Objective ans Solution

The goal is to find a trajectory which respects ATC restrictions and for which the total costs are minimal. The optimal solution to our problem is a trajectory from the departure airport to destination airport that is valid w.r.t. the graph, restrictions and aircraft performance while minimizing the total cost.

3.3 Simplifications

In this work, we are applying the following simplifications.

3.3.1 Navigation

We are going to use a generic cruise table 3.1 which allows for cruising on all 1000 feet regardless of the direction. The generality of the problem is not affected by this simplification.

Course from	to	True/Magnetic	Altitude from	to	Vertical Separation
0°	360°	T	0 ft	∞ ft	1000 ft

Table 3.1: Generic cruise table used for computation

3.3.2 Restrictions

For checking if an ATC restriction is active, we would have to check the actual time the aircraft is at the restricted element. Since we are using the departure time instead, this can lead to false results. Either we are avoiding a restriction that is not active anymore, or we are considering a restriction not active and violating it. The result can be that ATC will reject the found trajcetory.

In this work, we do not consider ATC restrictions of the “mandatory” form. They are constructed similar to the “forbid” restrictions, but they denote that something is

mandatory if a condition is met. However, since restrictions of the mandatory form can be translated into forbid restrictions, the generality of the problem is not affected.

3.4 Complexity

We are going to show the NP-hardness of the FPP by reduction from the problem of a path avoiding forbidden pairs (FAFP). The FAFP consists of a Graph G = (V,E), two fixed vertices s,t ∈ V and a set F of pairs of vertices $\in V$. The goal is to find a path from s to t, where at most one vertex of every forbidden pair of F is in the path, or to say with certainty that such a path does not exist. A forbidden pair (A, B) with $A, B \in V$ effectively allows to either use A, B but never both of them. The decision variant of this problem was shown to be NP-complete in [25].

Theorem 1 *The FPP is NP-hard.*

Proof 1 *Given a FAFP instance we are going to create an FPP problem as follows: The graph G will be used without any alterations and Dep_{Apt} will be s while Dst_{Apt} will be t. The restrictions C are constructed by creating two point crossing restrictions for every forbidden pair as shown in Equation 3.13.*

$$A \text{ closed if: } B \wedge B \text{ closed if: } A \qquad \text{for all } (A, B) \in F \tag{3.13}$$

The validity of every restriction is equal to the departure time s.t. all restrictions are active for the calculation. This already concludes the main part of the reduction, the rest is only to complete the FPP instance.

For simplicity let the weather function always return zero wind, and no deviation to ISA conditions for all altitudes. The APM always returns fuel-burn, altitude and duration zero, for all possible inputs. Because the altitude is always zero, the aircraft is cruising from departure to the destination on an altitude of zero feet. The cruise table and segment minimum and maximum altitude are constructed s.t. cruising is allowed at zero feet. Since the duration for traversing an arc is always zero the Cost Index can be ignored.

A solution to the initial PAFP exists iff there is a solution to the constructed FPP, hence the FPP is NP-hard.

CHAPTER 4

Reference system

In this chapter, we are going to explain the functional principle of an implementation that is used to set the baseline for comparing our approach to. It is based on Dijkstra's algorithm for finding the shortest path in the airway system. In this way, it will first find the route with the least cost wihout considering constraints. Next, a repeated process is applied to handle constraints, like the restrictions in the airway network. Although this method provides exact solutions to the problem, its runtime is already barely sufficient for practial use. Additionally instances get more and more complicated, while the industry aims to further extend the influencing factors.

4.1 Dijkstra's algorithm

Dijkstra's algorithm, published by Edsger W. Dijkstra in 1959, is a greedy algorithm for the single source shortest path problem in graphs with non-negative arc weights [3].

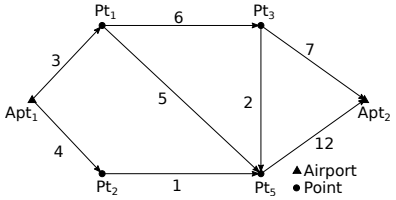
Dijkstra's shortest path algorithm 4.1 is a greedy algorithm that always expands the cheapest vertex of a so-called open list. *Dist* stores the distance from the source vertex s to each already explored vertex. In the beginning, the only available information is the distance to itself which is 0. To start, s will be put on to the open list, being its only element at that time. Next, a loop will always take the vertex u from the open list for which $distance(u)$ is the lowest among all vertices in the open list H. Now, for all outgoing arcs $(u, v) \in A$ of u we consider going to v via this arcs. If the cost to v via u using arc (u, v) is cheaper than the already known path to u (i.e. $dist(u) + l(u, v) \leq dist(v)$) we update it. If v has not been visited so far its distance will be infinite, and $dist(u) + l(u, v)$ will be trivially less than that. Because the arc weights are required to be non-negative, once a vertex is taken from the queue the lowest cost from s to it are known, and we can be confident that there will not be shorter paths to it discovered later. Once t is taken from the open list H we, therefore, know the shortest path from s to t is found.

Algorithm 4.1: Dijkstra's algorithm

Input: Graph $G = (V, A)$, $len(u, v)$ positive arc lengths $(u, v) \in A$, vertex $s \in V$
Output: Distances from vertex s to all other vertices

```
1  for v ∈ V do
2  |  dist(v) ← ∞
3  end
4  dist(s) ← 0
5  parent(s) ← s
6  H ← makequeue(V)                  // generate queue ordered by dist
7  while H ≠ ∅ do
8  |  u ← pop_min(H)                 // remove item with lowest distance
9  |  for (u, v) ∈ A do
10 |  |  if dist(u) + len(u, v) < dist(v) then
11 |  |  |  dist(v) ← dist(u) + l(u, v)   // set new discovered distance
12 |  |  |  parent(v) ← u
13 |  |  |  decreasekey(H, v)             // update key v to lower value
14 |  |  end
15 |  end
16 end
17 return dist, parent
```

By repeatedly following the *parent* vertex, starting from the target vertex t, we will eventually end up at s. This sequence is the shortest path from s to t in reversed order.



(a) Sample graph for explaination

	distance						parent
Apt_1	[0]						
Pt_1	∞	[3]					Apt_1
Pt_2	∞	4	[4]				Apt_1
Pt_3	∞	∞	9	9	[9]		Pt_1
Pt_5	∞	∞	8	[5]			~~Pt_1~~ Pt_2
Apt_2	∞	∞	∞	∞	17	16	~~Pt_5~~ Pt_3

(b) Result of Dijkstra's algorithm

Figure 4.1: Application of Dijkstra's algorithm to find a shortest path

Example 4.1 *In Figure 4.1 the application of Dijkstra's algorithm is shown to find the shortest path from vertex Apt_1 to Apt2 in the graph shown in 4.1a. Table 4.1b shows the final state of the data structure used in the algorithm. Since the cost of the target vertex Apt_2 is less than ∞ this means that there was a path found. To retrieve the path from the data structure we finally have to follow each consecutive parent starting from the target vertex. The result is the shortest path from the starting vertex to the target*

vertex in reversed order. In this example, we will obtain $Apt_1 \to Pt_1 \to Pt_3 \to Apt_2$ *with a total cost of 16.*

4.2 Application to flight planning

In order to use Dijkstra's algorithm for flight, planning we have to define a graph. In this case, we will be using a 3-dimensional graph $G(V_{3d}, A_{3d})$ defined by:

$$
\begin{aligned}
Altitudes &:= \{0, 1000, 2000, \ldots, 40000\} \\
V_{3d} &:= \{v_{alt} \mid v \in V, alt \in Altitudes\} \\
A_{3d} &:= \{(u_{alt_1}, v_{alt_2}) \mid (u, v) \in A, (alt_1, alt_2) \in Altitudes \times Altitudes\}
\end{aligned}
$$

For this algorithm, it does not matter if the vertices are in the plane or the 3-dimensional space. Therefore we do not need to extend it to include the altitude. We can extend the graph such that instead for every vertex $u \in V$ multiple vertices have the same geographic location but different altitudes.

In the case of flight planning, we want to reduce the total cost of traveling from the departure airport to the destination airport. For this thesis, this will only be the cost of fuel required. Therefore the optimization target changes from simply arc length (Algorithm 4.1) to the accumulated cost down a path. In Algorithm 4.2 we see a simple modification of Dijkstra that uses an APM to determine the fuel used for traversing an arc. The required parameters needed by the APM are the mass of the aircraft ($mass$) and start altitude (encoded in u) and target altitude (encoded in v). From the APM we expect here that if the aircraft cannot fly from u to v because of the given altitudes of the vertices and the current gross mass of the aircraft, the required fuel for traversal it

will return will be infinite. This way the vertex v will not be updated.

Algorithm 4.2: Flight planning algorithm based on Dijkstra

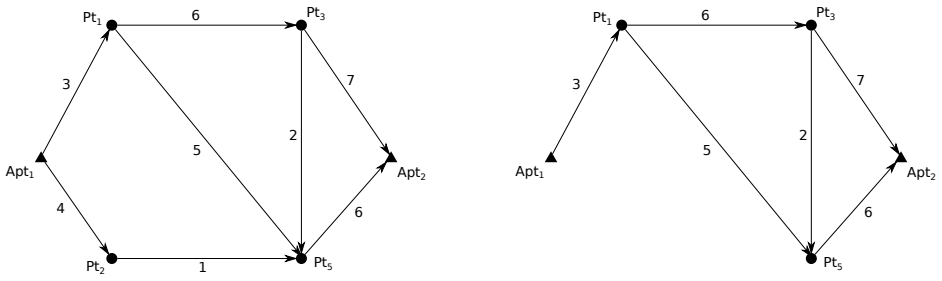
Input: Graph $G = (V, A)$, $len(u, v)$ positive arc lengths $(u, v) \in A$, vertex $s \in V$

Output: Distances from vertex s to all other vertices

```
1  for v ∈ V do
2  |  cost(v) ← ∞
3  end
4  cost(s) ← 0
5  time(s) ← T_departure
6  gm(s) ← M_takeoff                    // initial mass of the aircraft
7  H ← makequeue(V)
8  while H ≠ ∅ do
9  |  u ← pop_min(H)
10 |  for (u, v) ∈ A do
   |  |  /* using Weather model determine current weather on
   |  |     arc                                                  */
11 |  |  weather ← WF(time(u), mid_{u,v})
   |  |  /* using APM to determine fuel consumption              */
12 |  |  perf ← APM(gm(u), len(u, v))
13 |  |  cost_to_v ← cost(u) + perf_fuel + perf_duration · CI
14 |  |  if cost_to_v < cost(v) then
15 |  |  |  cost(v) ← cost_to_v
16 |  |  |  gm(v) ← gm(u) − perf_fuel
17 |  |  |  time(v) ← time(u) + perf_duration
18 |  |  |  decreasekey(H, v)
19 |  |  end
20 |  end
21 end
22 return dist
```

4.3 Extension for Restrictions

The previously explained algorithm does not consider restrictions of any form. Therefore, we are going to present a process that allows the integration of certain constraints, like ATC restrictions as specified in Section 2.2.8, in the problem.



(a) Graph specified by navigation data (b) Reduced graph

Figure 4.2: Respecting restrictions by changeing the graph

Consider the graph from Figure 4.2a for the following example, where Apt_1 is the departure, and Apt_2 is the destination airport. Additionally, we are going to add the following two constraints to the instance:

- $C_0 := Pt_2$ closed if $\mathsf{or}(Departure\_Airport\ Apt_1,\ Departure\_Airport\ Apt_5)$
- $C_1 := Pt_3$ closed if $\mathsf{not}(Point\_crossing\ Pt_5)$

First, let us have a look at the constraint C_0. The condition of C_0 can be evaluated solely based on information specified in the instance of the FPP because its only terminals are of the type *DepartureAirport* which is part of the instance. Because Apt_1 is for a fact the departure airport for this instance, the constraint tells us that Pt_2 must not be in the resulting trajectory. This can be achieved by removing Pt_2 and all its connected segments from the graph. In Figure 4.2b we see the graph after we removed Pt_2. It is obvious that the constraint C_0 can not be violated in the generated graph. Since any possible trajectory using Pt_2 would have been invalid, we know that we did not remove any valid solutions.

In the process that we present, it is best to integrate as many restrictions in the graph as possible. Therefore, we are using a three-valued logic which allows to increase the number of restrictions that we can already handle this way. Terminals like *Point crossing*, *Segment crossing* and *Airspace crossing* are treated as "unknown", while all the other terminals can be evaluated to true and false.

Example 4.2 *Consider the following constraint*
Pt_2 closed if or(*Departure Airport Apt_1, Point crossing Pt_1) Because for the given instance Apt_1 is the departure airport, even though we do not know if the trajectory will cross the point Pt_1 we can perform the following assertion:*
$Instance \models or(Departure\_Airport\ Apt_1,\ Point\_crossing\ Pt_1)$
$Instance \models or(True,\ Unkonwn)$
$Instance \models True$
Since it evalutes to true, we know we must not use Pt_2 which we can ensure by adapting the graph accordingly.

To integrate as many restrictions as possible in the graph we evaluate all restrictions using the information given by the instance. Based on the resulting truth value we proceed with the following actions:

- true ⇒ remove the corresponding element from the graph
- false ⇒ the constraint will not be violated and can, therefore, be ignored
- unknown ⇒ handle the constraint later

We call the result of this process the *reduced graph* because it is prepared in a way s.t. the consideration of constraints reduced the search-space.

If we continue our current example, C_0's condition evaluates to true which caused us to close Pt_2. Since C_1's condition evaluates to "unknown", the graph that we constructed (Figure 4.2b) is already the reduced graph of the instance. We can now use Algorithm 4.2 on the reduced graph without ever violating C_0. The result that we get for applying the algorithm to the reduced graph is $t_1 := Apt_1 \rightarrow Pt_1 \rightarrow Pt_3 \rightarrow Apt_2$.

As already mentioned before, now that we have a complete trajectory, we have to validate all previously unhandled constraint. If the trajectory does not violate any restrictions, we can safely say that this is the cheapest trajectory respecting all restrictions. However, if it violates any restriction we have to proceed.

Because $t_1 \not\models C_1$, t_1 is not a valid route for the given problem. The reason for the violation of the constraint is that while $Pt_3 \in t_1$ also $Pt_5 \notin t_1$. In order to respect this constraint we have to run further optimizations for which we ensure that this restriction is not violated anymore. Because we want to maintain optimality of the result and there are several ways how to avoid violating the constraint, we have to run several optimizations for which we each add a minimal set of constraints that ensure just that.

First off, we can avoid the violation by merely avoiding the element of the restriction altogether. Second, we have to find all possible cases that make the condition false which we achieve by using Algorithm 4.3.

Algorithm 4.3 recursively traverses a given condition-tree. The result is a set of the form $\{reopt_0, reopt_1, \ldots, reopt_n\}$, where $reopt_i$ might look like this $\{Pt_1, \neg Airspace_3\}$. Each

Algorithm 4.3: Generating re-optimization constraints

Input: Condition C, *negate* if subtree is negated
Output: Set of reoptimizations

```
1  if C is terminal then
2  |  if negated then
3  |  |  return {{element}}                    // element must be visited
4  |  else
5  |  |  return {{¬element}}                   // element must be closed
6  else if C is operator Not then
7  |  return rec(child, ! negate)                          // flip negation
8  else if (C is operator And) ≠ negate then
   |  /* Operation similar to cross product, but empty sets
   |     are ignored                                                  */
9  |  reoptimizations ← ∅
10 |  foreach c ∈ children do
11 |  |  sub ← rec(c, negate)
12 |  |  if reoptimizations = ∅ or sub = ∅ then
13 |  |  |  reoptimizations ← reoptimizations ∪ sub        // Ignore empty
14 |  |  else
15 |  |  |  reoptimizations ← reoptimizations × sub       // Cross product
16 |  |  end
17 |  end
18 |  return reoptimizations
19 else if (C is operator Or) ≠ negate then
   |  /* Append all reoptimizations                                  */
20 |  reoptimizations ← ∅
21 |  foreach c ∈ children do
22 |  |  reoptimizations ← reoptimizations ∪ rec(c, negate)
23 |  end
24 |  return reoptimizations
25 end
```

set represents an individual re-optimization and the elements indicate the way how the restriction is avoided. A negated element requires us to close it in the graph, while an original one requires us to find a solution that uses this element.

Example 4.3 *In order to respect or(Point_crossing A, Point_crossing B) we have to avoid both points A and B at the same time, represented as* $\{\{\neg A, \neg B\}\}$.

Example 4.4 *In order to respect and(Point_crossing A, Point_crossing B), we could avoid both points at the same time, but since it is enough to avoid just one we need to do so to maintain optimality. Because we do not know which one to choose, we have to run two individual optimizations represented as* $\{\{\neg A\}, \{\neg B\}\}$.

Algorithm 4.3 applies the pattern from Example 4.3 and 4.4 and also uses De Morgan's law s.t. negations are pushed down to the terminals. The sequence operator is left out for simplicity, but it can be treated like the *and* operator with the additional condition ensuring the conservation of the order in which elements are visited.

To continue the running example, we obtain $\{\{Pt_5\}\}$ by using Algorithm 4.3 on the condition of C_1. Combining it by avoiding the element Pt_3 we get the necessary optimizations:

1. try to make Pt_5 part of the path
2. try to avoid Pt_3 altogether

For the first one $\{Pt_5\}$ we have to find a path containing Pt_5. We can use our FPP algorithm to find the cheapest trajectory Pt_5 and subsequent the cheapest path from Pt_5 to Apt_2.

$$t_2 := Apt_1 \to Pt_1 \to Pt_5 \oplus Pt_5 \to Apt_2 \tag{4.1}$$
$$t_2 := Apt_1 \to Pt_1 \to Pt_5 \to Apt_2 \tag{4.2}$$

For the second case $\{\neg Pt_3\}$ we must find a path that avoids Pt_3. Therefore, we again have to use the reduced graph and additionally remove the vertex Pt_3 before performing an optimization.

The result we obtain is

$$t_3 := Apt_1 \to Pt_1 \to Pt_5 \to Apt_2 \tag{4.3}$$

In our example, none of the gathered trajectories (t_2 and t_3) violate any additional restrictions. If any of them had violated a restriction, we would have to repeat this process and combine all the constraints generated by Algorithm 4.3 with the constraints that have been used in the optimization leading us to this point.

In this scenario, all consequently obtained trajectories were the same, but this must not be the case in general. The result that we are looking for is the cheapest, valid route, which will be the one that we are going to return. Next, we will formalize the previously shown steps in a generalized process that finds the cheapest trajectory in the presence of constraints.

Please note that, although the condition terminals may contain an additional altitude range to which they apply (see Table 2.6), it was left out of this section in order to increase readability and to keep the focus on the main points. It should be enough to say that it is just an additional attribute of the terminal that needs to be passed along the way and respected during optimizations.

General process

In Figure 4.3 we can see the overall schema of the process. Please note that we depicted this recursive process in an iterative manner. We start by building a graph reflecting the airway system. All constraints that can already be decided by the parameters of the FPP instance, like depature and destination airport get applied, resulting in the reduced graph. This graph is then used by Algorithm 4.2 to find the cheapest trajectory, without considering any additional restrictions. Next, we have to evaluate the resulting trajectory to see if it violates any of those restrictions that where undecidable beforehand. If this is the case, we build a set of constraints which will prevent violating this restriction again, we will call them *re-optimization constraints*. In the case that this trajectory was the result of an optimization that already contained re-optimization constraints, we have to combine them with those newly generated ones. All of the possible cases that ensure avoiding this restriction will be queued for later optimization. If the trajectory did not violate any restrictions, i.e. , was valid, we store it as a potential final result. As long as there are re-optimization constraints in the queue, we take any of them and repeat this process. Once the queue is empty, we are done. If not a single optimization resulted in a valid route, we can conclude that this instance of the FPP has no solution. In real life, this is a very unlikely case and probably raises the question of wrongly entered restrictions by the ATC. On the other hand, if there has been at least one trajectory that was valid, we are going to return the one with the least cost as the final solution to the FPP.

Pruning

The runtime of this method heavily depends on the number and the structure of the ATC restrictions. Although to try to reduce the number of individual optimization runs by taking as many restrictions into account by adapting the graph, this is not always possible. Currently, there are more than 20.000 ATC restrictions with bigger conditions than we showed in our examples. Most of the restrictions require several reoptimizations, some of which can easily be discarded. If the reoptimization constraint is contradicting like $\{\ldots, Pt_i, \ldots, \neg Pt_i, \ldots\}$ we can reject this optimization because it can not result in

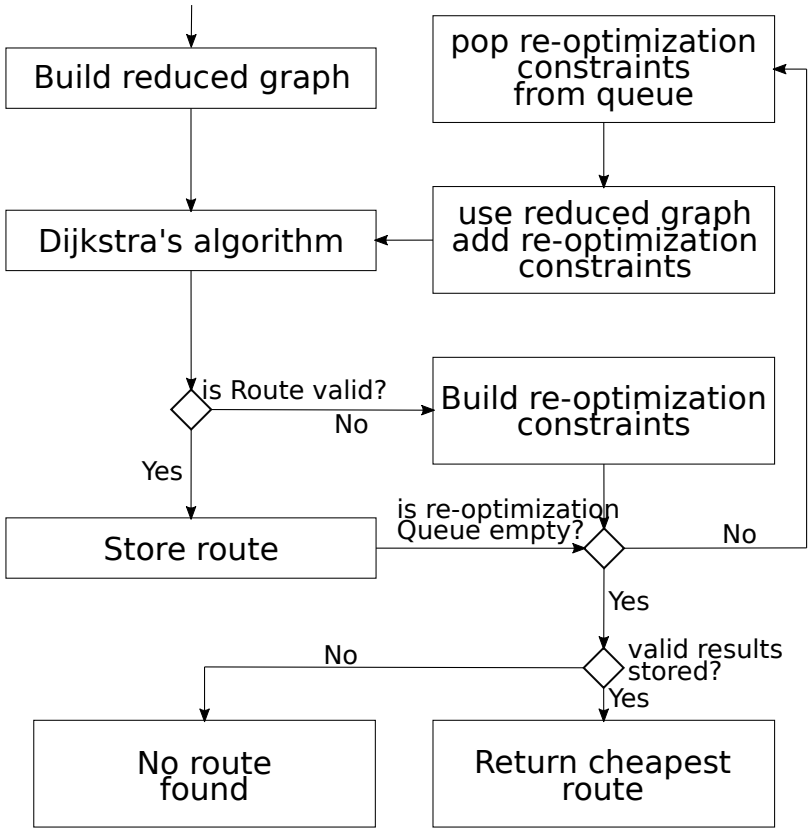


Figure 4.3: Process of handling constraints in the graph

a trajectory. Similar to the previous example, we could be required to use a point that is part of an airspace that we are required to avoid.

Another critical measure we took is to prune entire branches. Consider a trajectory that is violating a restriction, although it is invalid we have a corresponding cost. All subsequent optimizations that we are going to perform in order to make it valid, will only result in trajectories at a higher cost. The reason is that we are adding an additional constraint. Therefore, every time we obtain a valid trajectory, we can prune all optimizations that are currently queued and have a higher bound. Furthermore, it is not necessary to queue any re-optimizations from an invalid trajectory if we already found a solution that has a lower cost.

In Figure 4.4 can see an example of a re-optimization process. The root node represents the first optimization without any additional constraint $\{\}$ which resulted in an invalid trajectory with a cost of 1800. From the root node, we have two necessary re-optimizations, to the left with $\{A\}$ (A closed) and $\{B\}$ (B closed) to the right. We picked the left one first, again resulting in an invalid trajectory, with a cost of 1950, and subsequently $\{B\}$. Finally, $\{A, C\}$ returned a valid route with a cost of 1960. This means that we can now

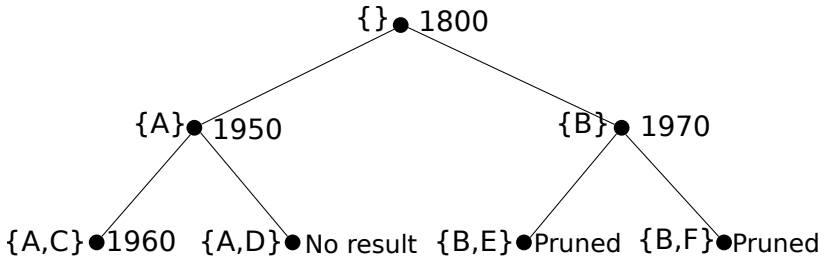


Figure 4.4: Example re-optimization tree

prune all re-optimizations with a higher bound than 1960, which are all re-optimizations caused by $\{B\}$ (1970) in this example. Next, we still have to optimize $\{A, D\}$ because it could lead to a better result. As it turned out, we did not find a result using these constraints, which leaves us with a total of 5 optimizations. Since we do not know what the result of the two re-optimizations that we pruned was going to be, we can not tell what the total number of saved re-optimizations was, but it was at least two.

CHAPTER 5

Proposed heuristic approach

The optimal trajectory for given departure and destination airports depends on many parameters. However, within a certain range, best trajectories do only change marginally in dependence of parameters like the aircraft's takeoff weight. When considering changes in the weather, navigation data and restriction activation due to the departure time, we expect that there is only a limited set of favorable trajectories. Based on this assumptions, the idea is to take already known trajectories from previous optimizations and those from past flight plans, combine and adapt them to get new ones that are the best fit for the current situation. This way we can also profit from previous calculations and take advantage of the fact that the airway network does not completely change from one optimization to the next. The best case scenario is that already known trajectories, from previous calculations, are still valid and can be reused.

5.1 Genetic algorithms

A *genetic algorithm* (GA) is a metaheuristic inspired by Charles Dawrin's process of natural selection. It was developed by John Holland in 1960 as an effort to bring nature's way of evolution to computational optimization [26].

As previously mentioned in Section 1.2, genetic algorithms have already been used in the field of flight planning. In [13] a genetic algorithm was used to optimize a flight from TOC to TOD on a single fixed altitude and without considering ATC restrictions. In [12], a GA was used to find an improved solution in a 3D grid that was created around a pre-planned trajectory, not considering ATC restrictions. A genetic approach was chosen because of its low computational effort, such that it can be carried out on an onboard computer. In [27], a GA was used to solve the k-shortest path problem.

Genetic algorithms belong to the category of population-based algorithms, which means that they operate on a set of candidate solutions, called the *population*. A solution of

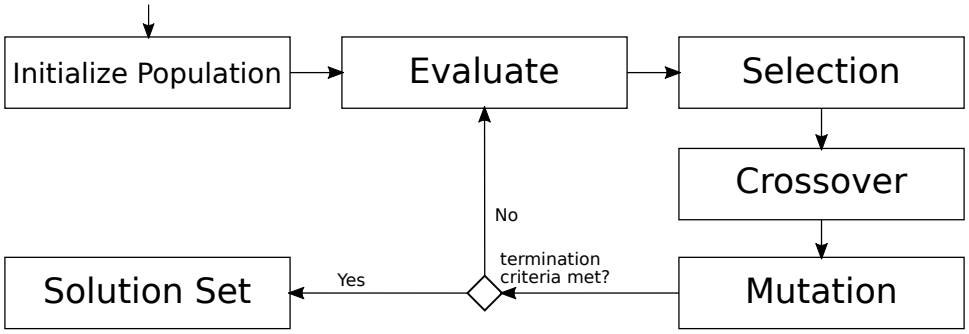


Figure 5.1: Canonical genetic allgorithm processes

the population is called an *individual*. In Figure 5.1, as well in more detail in Algorithm 5.1 we can see the general architecture and the building blocks of a genetic algorithm. Starting with an *initial population*, each individual gets evaluated using a *fitness function* to determine the quality of each individual. Next, a set of individuals is selected from the previous population to move on and make up a new generation. Parent individuals get combined to create new individuals in the *crossover* phase. Random changes called *mutations* are made to the individuals to increase diversity. Finally, we have to decide if we are satisfied with the result, or if we want to repeat this process. In the following sections, we are going to explain each phase in more detail.

Algorithm 5.1: Canonical GA

```
1  t ← 0
2  initialize(P_t)
3  evaluate(P_t)
4  while not termination-condition do
5  |  t ← t + 1
6  |  Q_t^s ← select(P_{t−1})
7  |  Q_t^p ← crossover(Q_t^s)
8  |  Q_t^m ← mutate(Q_t^r)
9  |  P_t ← Q_t^m
10 |  evaluate(P_t)
11 end
```

5.1.1 Fitness

The *fitness* of an individual is a measure of how well it performs. It is used to compare individuals to find the best ones. For some problems, it is beneficial, or even necessary, to allow invalid solutions. In such cases, a common way to handle those is to add a penalty to solutions that are invalid. This penalty changes the fitness of an individual such that valid ones are preferred over invalid ones.

5.1.2 Initial population

As already mentioned before, genetic algorithms fall in the category of population-based algorithms. All the following phases are based on the assumption that there is a preexisting set of solutions. For the future efficiency of the algorithm, it is essential that the initial solution already contains a high degree of diversity. One way to get a diverse population is by using a randomized heuristic to creating initial individuals.

5.1.3 Selection

In the *selection* phase, a set of individuals get, as the name suggests, selected for the next generation. The fitness of an individual should be reflected in its probability for getting selected. There are a variety of methods for selecting individuals from a population. In the following, we are going to show two popular selection methods. Both of these methods return a single individual per invocation. If a total of n individuals are desired to be selected for the next population, the methods need to be used n-times. This process simulates the survival of the fittest individuals.

Roulette wheel selection

In the *roulette wheel* selection method, every individual gets a probability proportional to its fitness. If an individuals fitness is as high, as the sum of the fitness of all other individuals, there will be a 50% chance it will be selected. Algorithm 5.2 shows an implementation of this approach. First, the total fitness of all individuals needs to be calculated. The next step is to guess a number between zero and the total fitness randomly. This process is comparable to throwing the ball into a roulette game, where every field represents an individual. In contrast to the popular gambling game, not all fields have the same size, but their size is proportional to the fitness of the individual it represents. Finally, we have to find out on which individual the ball has fallen, i.e., was randomly selected. Performing a selection like that has an inherent problem which is already indicated in the mentioned example. Depending on the range of the objective function an individual with much higher fitness than the average could dominate all the other individuals. Therefore all other solutions get lost which results in a population of mostly inbreds of this solution. On the other hand, if the range of fitness values is small, one might end up with a purely random selection. Therefore fitness scaling is crucial to get the right balance between exploration and exploitation. [28]

Tournament selection

The *tournament selection* is the second procedure to select an individual from a population that we are going to show. As in Algorithm 5.3, a fixed number of individuals, the tournament size k, are randomly picked from the population. From all of those selected individuals, the one with the best fitness function wins the tournament, i.e. , gets selected. The tournament size allows adjusting the pressure on the individuals. Compared to other selection methods, the relative difference of the fitness measure does not influence the

Algorithm 5.2: Roulette wheel selection

Input: Population p
Output: Select individual randomly based on fitness

```
1  total ← 0
2  foreach individual ∈ p do
3  |  total ← total + f(individual)
4  end
5  guess ←random [0, total]            // pick value between 0 and total
6  foreach individual ∈ p do
7  |  guess ← guess − f(individual)
8  |  if guess < 0 then
9  |  |  return individual
10 |  end
11 end
```

outcome. If the tournament size is reduced, individuals with lower fitness have a higher chance to be selected then compared to a higher tournament size where the probability that a better individual makes it into the tournament is consequently higher.

Algorithm 5.3: Tournament selection

Input: Population p, Tournament size k
Output: The best individual from k randomly selected ones

```
1  winner ←random p                          // pick random individual
2  while k > 1 do
3  |  individual ←random p
4  |  if f(individual) > f(winner) then
5  |  |  winner ← individual      // assume higher fitness is better
6  |  end
7  |  k ← k − 1
8  end
9  return winner
```

5.1.4 Crossover

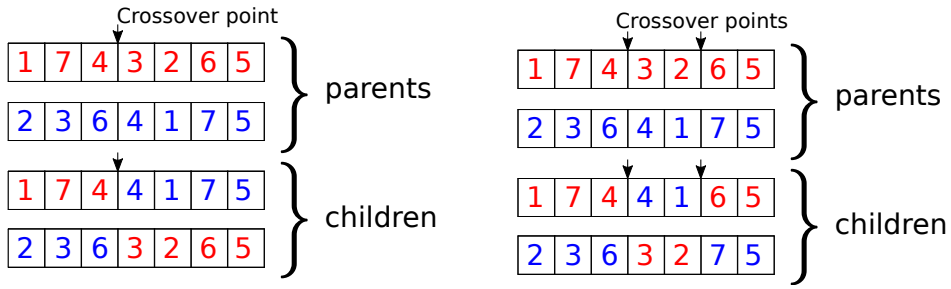
In this phase, called *crossover*, parent individuals are selected to be combined in order to create new individuals (also referred to as recombination). Using a so-called *crossover probability*, we can control how likely it is for two individuals to be combined to generate an individual. There are several well-established crossover procedures of which we are going to explain three basic ones in this section. The selection of the right crossover operation is dependent on the problem and the encoding of the individuals.

Single point crossover

Given two individuals, a single point at each is selected at which they get split put and be put together with a piece of the other individual. Figure 5.2a shows an example of a single point crossover on two individuals.

Two point crossover

The two point crossover is similar to the single point crossover, but this time two points get selected per individual and the middle part gets exchanged. In Figure 5.2b we can see the generation of two new individuals using two point crossover. The individual, in this example, is an array of seven integers in the range of one to seven. Depending on how the individual is build and what it represents, this type of crossover might result in an invalid solution.



(a) Single point crossover (b) Two point crossover

Figure 5.2: Examples for single and two point crossover

Cycle crossover

If the individual is, for example, representing an ordering of some sort, then the *cycle crossover* is a method that allows exchange without ever generating an invalid individual. The previously presented methods do not preserve the set of values used in the individuals. For example in the single point crossover, one child ended up with two fours, but without a three. Instead of merely exchanging parts of the individual at a fixed position, we are looking for a cycle of values across both parents. Then the individual cycles get transferred to the child individuals either like they were found, or swapped. This way only a set of values, gets exchanged with the same set of values from the other individual.

5.1.5 Mutation

A problem in many optimization methods is getting stuck in local optima. The *mutation operation* is an attempt at escaping local optima, by applying random changes to individuals. Doing so increases the diversity of the population which makes the optimization explore more of the search space. This change can be as simple as flipping a bit, but

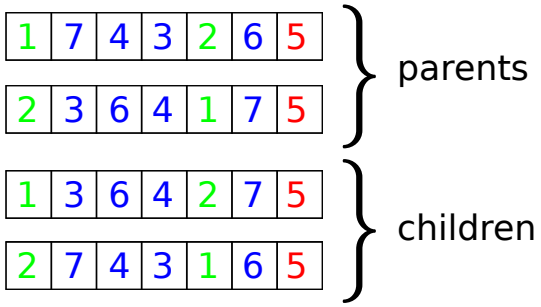


Figure 5.3: Example of a cycle crossover

again, this depends on the solution representation. The *mutation probability* defines how likely it is for an individual to get mutated.

5.1.6 Termination criterion

The final piece of the puzzle is the *termination criterion*. One big problem when using a genetic algorithm is that it is usually not known when the optimal result is found. Therefore, we need to define a point at which we decide to stop the search for a better solution. Common termination criteria are:

- number of iterations
- certain fitness reached
- number of iterations without improvement
- when the improvement rate declines
- amount of time spent

Also, a combination of multiple criteria can be used.

5.2 Solution representation

A solution to our problem at hand, the FPP, is a trajectory. Therefore we need to find a suitable representation of a trajectory for use with a genetic algorithm. In order to do so, we have to encode the path as well as the altitude profile. The path will be represented by a sequence of consecutive arcs. This also means that the solution will not be of fixed length, but variable depending on the number of arcs used. For the altitude profile, we considered relative altitude changes and absolute target altitudes. When using relative altitude changes, a combination of two individuals can lead to the situation that not the same altitude that was climbed was also descended. This can lead to the fact that either at the destination we are still in the air, or hit the ground before. Also, it is not directly

noticeable if the current altitude is within the bounds of the segment, or if the aircraft cruises on an incorrect altitude. Simply combining two individuals using single point crossover (assuming these two individuals share a waypoint), can cause the previously mentioned problems.

Few of these problems cannot arise when using absolute target altitudes of the segment. It can easily be checked if the target altitude is within the bounds of a segment and also if it aligns with the cruise altitude. For these reasons, we decided to go with absolute target altitudes. The resulting representation is shown in Figure 5.4, where the target altitude is represented as a blue line.

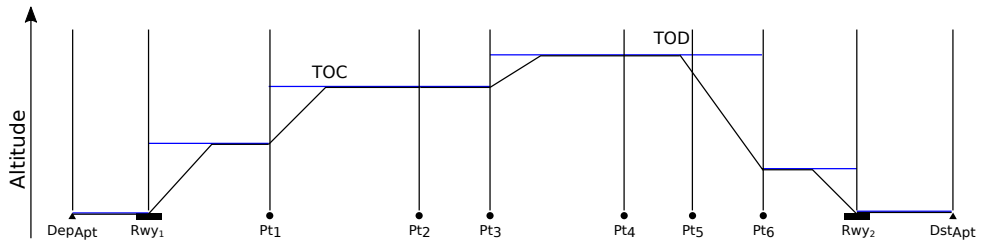


Figure 5.4: Representation of a trajectory, target altitude in blue

One disadvantage that we identified when using absolute altitudes is that not all changes to the individual affect the resulting trajectory. Consider Figure 5.5, with two different sequences of target altitudes (in blue and in red) and the trajectory the aircraft performed in grey. The problem occurs on segments where there is a climb or a descend throughout the whole segment. Example 5.1 additionally shows the encoding using the blue target altitudes. Since the target altitude is never reached, it can be higher (in the case of a climb) or lower (in the case of a descend) without a change to the trajectory. Because we do not know when a climb or descend is finished just by looking at the individual, we cannot tell if the change affects the trajectory without the complete aircraft performance calculation. When using the relative altitude change representation, we would have had the same problem, even though in a slightly different form.

Example 5.1 *This example shows the encoding of the blue individual shown in Figure 5.5. It consinsts of a sequence of arcs with a correspnding target altitude.*

$[((P_1, P_2), 7000\ \textit{foot}), \ldots ((P_6, P_7), 7000\ \textit{foot}), ((P_7, P_8), 0\ \textit{foot}), ((P_8, P_9), 0\ \textit{foot})]$

Another problem is that we do not know when the final descend needs to be initiated such that the aircraft reaches the ground precisely at the runway. As we can see in Figure 5.4, the TOD is initiated during a segment and not exactly at a waypoint. In Figure 5.6 we can see again the same solution represented as in Figure 5.4. The red trajectory is the result of the aircraft following the target altitudes as they are defined. At the end of the trajectory, we can see that the aircraft reached the ground after the runway. To

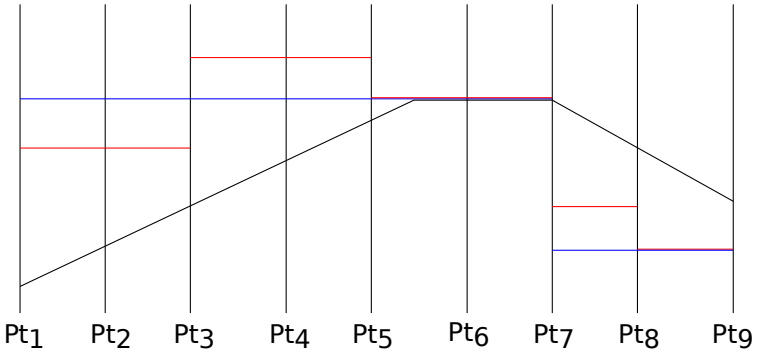


Figure 5.5: Representation of a trajectory, target altitude in blue

solve this problem, once the aircraft calculation is done as defined before, we are going to perform an aircraft performance calculation in reverse, starting from the destination (in blue). This reverse calculation is done until we cross the forward calculation, and the two trajectories get merged. The final trajectory, matching the one shown in 5.4, is shown in green.

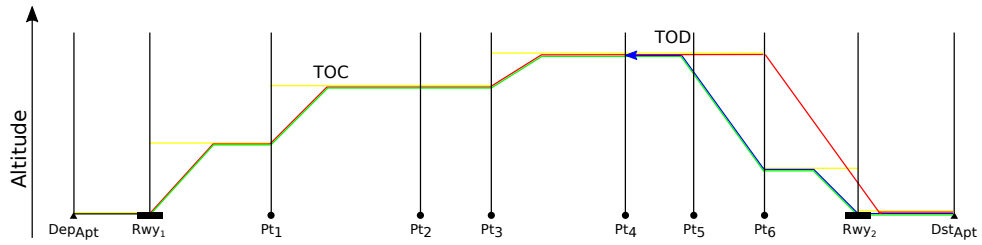


Figure 5.6: Finding the TOD and performing the final descent

5.3 Objective function

The objective is to find the cheapest trajectory, as specified in Section 3.1.1, while conforming to ATC restrictions. Therefore it is natural to use the total cost of the trajectory (consisting of primary and secondary cost) as the objective value that we want to minimize. Because our solution representation allows to model solutions that are not valid w.r.t. ATC restrictions, we need to add a penalty to such solutions.

The penalty that we propose is based on how deep vertically a trajectory is "in" a restriction. A restriction usually can be avoided by going above or below it. In the case where we cannot avoid violating a restriction without changing the path lateral, we set the penalty to infinity. Based on the condition, there could be a less costly way out than avoiding the restricted element itself. Therefore we consider all possible changes we can make, in order to avoid the violation and take the smallest necessary change. If a

trajectory violates various restrictions, we sum the necessary change for each restriction. It can be the case that multiple restrictions are avoided by the same change. However, to keep it simple, we only consider each violated restriction individually. This simplification might result in a trajectory being over-penalized. Since the objective function is called many times, it is essential that it can be calculated efficiently. The final result of this procedure is a value in feet, which we add 1:1 as a penalty in kg to the total cost of the trajectory. Example 5.2 and 5.3 show how the penalty is calculated for the violation of a simple restriction.

Example 5.2 *In Figure 5.7 we see a trajectory that crosses a restriction on the segment* (Pt_2, Pt_3)*. The restriction is from 18000 foot to 30000 foot. We are crossing the restriction at an altitude of 29000 foot, which is 1000 foot too low respectively 9000 foot too high to avoid it. The small deviation is 1000 foot, not considering if this is even possible to avoid. Therefore the penalty we add to this trajectory is 1000 kg.*

Example 5.3 *Let us reuse the previous example (Example 5.2) and additionally assume that,* Pt_3 *is closed completely (from the ground to unlimited) if* Pt_1 *was crossed. This means that just changing the altitude is not sufficient to ever make the trajectory valid. Therefore the penalty for this trajectory is infinite.*

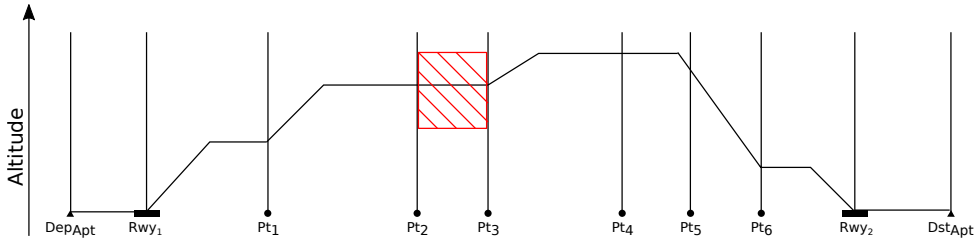


Figure 5.7: A trajectory crossing a restriction

In Equation 5.1 we formulate the deepness of a restriction that is violated. Let *Reoptimizations* of v be the set of possible ways to avoid violating a restriction generated by Algorithm 4.3 for the condition of the restriction v including avoiding of the element itself. Each of those ways to avoid that the restriction is violated consists of a set of constraints, for which $doge_{above}^{trajectory}$ indicates the necessary altitude change (by going higher) to avoid the violation. The same is possible by going lower (denoted by $doge_{below}^{trajectory}$) than the actual altitude, whatever is smaller is taken. All of those altitude changes get accumulated to get the total change in altitude for avoiding the restriction in this particular way. Since there are several ways how a restriction can be avoided, we take again the minimum of those.

$$deepness_v := \min_{\substack{Avoid \in \\ Reoptimizations(v)}} \sum_{\substack{constr \in \\ Avoidance}} \min \begin{pmatrix} doge_{above}^{trajectory}(constr), \\ doge_{below}^{trajectory}(constr) \end{pmatrix} \tag{5.1}$$

The final objective function is shown in Equation 5.2.

$$objective := \underbrace{total\ cost}_{\text{From Equation 3.12}} + \sum_{v \in Violations} deepness_v \tag{5.2}$$

5.4 Initial population

The initial population consists of solutions built from preferred routes provided by Eurocontrol for each city pair. Additionally, routes from all kind of sources can be integrated, like results from previous optimizations. Most airports are connected by multiple flights a day, by different operators. For all of these flights, a flight plan needs to be filed to ATC. Therefore there many past flight plans available. The strategies that we are going to explain for mutating a given solution can easily be changed to generate solutions. This approach can be used to generate an initial population for city pairs where no data is available. Because the available routes we have are less than the population size we choose, we use all available solutions for the initial population. If there are more initial solutions available, it would be best to start with a diverse set, rather than all similar solutions.

5.5 Selection

We choose the tournament selection method to pick individuals from the population. The decision was based on the fact that for roulette wheel selection, without a fitness scaling function, the relative objective is playing a much higher role. Furthermore, it is expected that the penalty that we add because of restriction violations has a higher influence then compared to the tournament selection. To counteract that, we have to find a suitable scaling, while the tournament selection can be used as is.

Because of the random element in the selection process, it is not certain that the fittest individual will always move to the next generation. This means that the best solution might get "lost" during the selection phase. As a result, the best solution found, is not necessarily the fittest individual of the last generation. Moreover, this has an impact on the whole optimization process. An extension to the GA called *elitism* always allows the best individual to move on to the next generation as it is.

5.6 Crossover operation

An inherent problem of our solution representation is that a meaningful combination of two individuals is difficult. If two trajectories cross perfectly, lateraly **and** verticaly, one or multiple times, it is easy to perform a single or double point crossover. However, given the size of the graph, this case is somewhat rare. A solution would be to add segments to the graph when necessary, keeping in mind that an individual using such a segment is not valid w.r.t. the graph and therefore not a valid solution to the FPP. The added segments for the crossover operation would require some penalty in the objective function, such

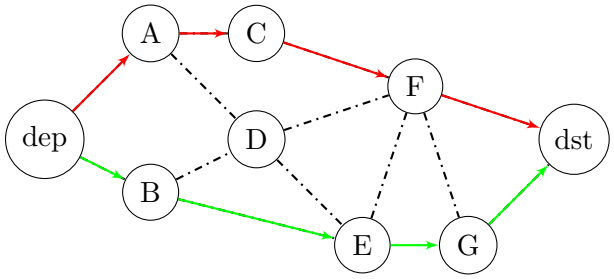
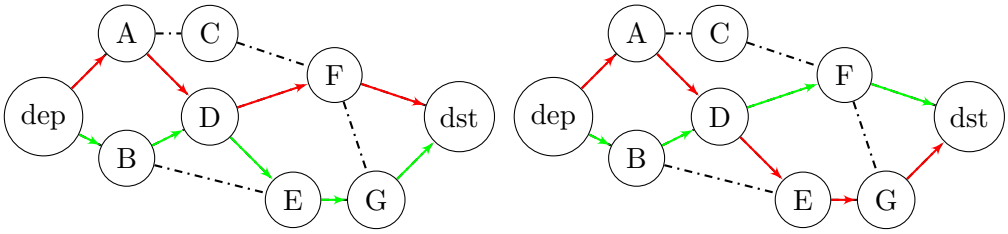


Figure 5.8: Two routes, no crossover possible

that individuals without them are superior. Finding a good value is difficult because these artificial segments would not have any restrictions, while regular ones do. So if all regular arcs have restrictions, it could still be cheaper to use the artificial ones. Because of this, we choose to use an approach where we are combining solutions that share at least one waypoint (i.e. cross lateraly) and perform single point crossover. If two individuals do not cross, they are incompatible for crossover (shown in Figure 5.8).

Figure 5.9 shows an example of two compatible lateral routes which can be used to generate two new solutions. D is a common waypoint of both routes, so it is possible to take the path from *dep* to D from one route and combine it with the path D to *dst* from the other route. Same can be done with the other halfs of the routes to generate a total of two new individuals. This might lead to loops in the path of a generated individual which need to be detected and removed.

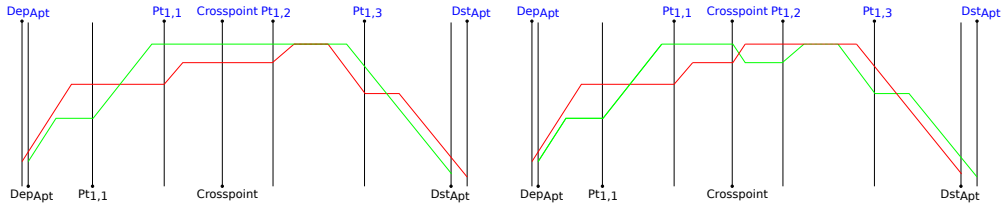


(a) Before crossover (b) After Crossover

Figure 5.9: Two individuals before and after crossover.

Figure 5.10 shows two trajectories from the vertical perspective, centered around a common waypoint *crosspoint*. Even though they start and end at the same waypoint, they do not line up, because the two trajectories have different ground distances. This visualization is chosen to show how the crossover is performed vertically.

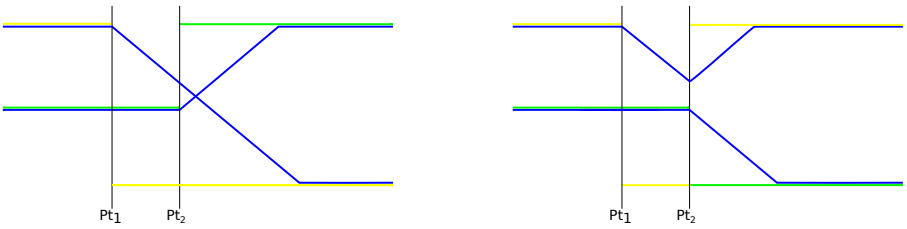
A problem in the trajectory that might arise due to the crossover is shown in Figure 5.11. The yellow and green lines show the target altitude of two trajectories (Figure 5.11a) before the crossover operation is performed at the waypoint *crossover*. While the trajectory generated by the green target altitude lines is perfectly fine, the other one is not (Figure 5.11b). The trajectory generated by the yellow altitude lines performed



(a) Before crossover (b) After Crossover

Figure 5.10: Two individuals, in the vertical perspective, before and after crossover.

a descend, followed by a climb without a cruise in between. This same problem can also arise oppositely, i.e., a climb directly followed by a descent. A trajectory containing such a spike will not be accepted by any pilot or ATC. Just by looking at the individual, without using the aircraft performance model, we cannot tell the if target altitude is reached in the given distance before it changes again. We solved this problem with an addition to the objective function. All individual waypoints, for which the current altitude is not an allowed cruising altitude, are inspected and checked if such a case is present (i.e., the altitude before and after are both higher (or lower)). For every point that matches this description, we add a small penalty of 100 kg to the objective function, and we mark the solution as invalid.



(a) Before crossover (b) After Crossover

Figure 5.11: Problem of climb after descend because of crossover.

5.7 Mutation operation

Our solution representation consists of a sequence of arcs with a corresponding target altitude. Therefore we can change the path, as well as the target altitude. Next we are going to explain changes to the path and later in this section we are going to show mutation on the target altitude.

The change we are going to apply is to divert from the original path at a certain point and join it again at a later waypoint. Changing the patch is only possible starting at waypoints of the path that have at least two outgoing arcs. For most arcs (u, v) there also exists a counterpart (v, u) in the graph. Since we do not want to go back to a waypoint that we already visited, we cannot count these arcs for the decision. Based on these

criteria we are going to randomly pick one of these waypoints as the starting point for the mutation. Algorithm 5.5 shows this process in more detail.

Algorithm 5.4: Find starting point for mutation

Input: Graph $G = (V, A)$, Path P
Output: Mutated individual

```
1  choices ← ∅
2  visited ← ∅
3  for (u, v) ∈ P do
4  |  visited ← visited ∪ {u}
5  |  for (u, a) ∈ A do
6  |  |  if a ∉ visited and a ≠ v then
7  |  |  |  choices ← choices ∪ {u}
8  |  |  end
9  |  end
10 end
11 point ←Random— choices
12 return point
```

The next step is to randomly choose a different outgoing arc of the waypoint. As already said, we do not want to go back to an already visited waypoint, neither do we want to use the same arc as before. Again, we are using a random variable to decide which arc we are going to use. However, this time we do not want the probability to be uniform but take information about the arc into account. Choosing an arc that directly points towards the destination seems like a good decision while going in the opposite direction, towards the departure, will hardly result in an improved solution. Therefore we will define the most direct way possible for visiting a point in Equation 5.3 and call it $diversion_{min}$. Let the *diversion* on arc be the distance from the departure airport to its tail, its length, plus the distance from its head to the destination airport like shown in Equation 5.4. The maximum diversion that we consider is an additional detour of 15% of the direct distance $diversion_{min}$.

The probability to select an arc will be based on the diversion that is created by using it. Between a diversion of $diversion_{min}$ and $diversion_{max}$ we linearly interpolate from 0 to 1 respectively. All arcs with a diversion bigger than $diversion_{max}$ get a probability of zero to be selected. Preliminary experiments showed that a value of 15% was a good compromise between pushing into the correct direction while still allowing changes off the great circle line.

$$diversion_{min} := \text{distance}(Dep_{Apt}, u) + \text{distance}(u, Dest_{Apt}) \tag{5.3}$$

$$diversion := \text{distance}(Dep_{Apt}, u) + len_{(u,v)} + \text{distance}(v, Dest_{Apt}) \tag{5.4}$$

$$diversion_{max} := diversion_{min} \cdot 1.015 \tag{5.5}$$

Algorithm 5.5: Pick random out arc

Input: Graph $G = (V, A)$, Path P, (u, v) arc to replace
Output: Random outgoing arc

$exclude \leftarrow$ points up to u
$choices \leftarrow \emptyset$
$diversion_{min} \leftarrow$ `distance(`Dep_{Apt}`, u) + distance(u,` Dst_{Apt}`)`
$diversion_{max} \leftarrow diversion_{min} \cdot 1.015$
for $(u, v') \in A$ **do**
  $visited \leftarrow visited \cup \{u\}$
  if $v' \notin excluded$ *and* $v' \neq v$ **then**
    $diversion \leftarrow$ `distance(`Dep_{Apt}`, u) +`$len_{u,v'}$`+ distance(`v'`,` Dst_{Apt}`)`
    `/* Calculate probability for choosing this arc */`
    $prop \leftarrow$ `linear(`$diversion_{min}$`,` $diversion_{max}$`,` $diversion$`)`
    $choices \leftarrow choices \cup (prop, (u, v'))$
  end
end
`/* Randomly select arc from set of tuples, where the first element defines the probability to be picked */`
$arc \xleftarrow[Random]{weighted} choices$
return arc

When performing the mutation, we want to append a random outgoing arc to our trajectory repeatedly. How many steps of random appends we are going to perform will be a parameter to the mutation operation. Once these random steps are added, we need to join this partial trajectory with the original one. Therefore we are going to add random arcs to the end of the solution towards the closest waypoint of the original solution. This process will be performed repeatedly until the old trajectory is joined. It might as well join the old trajectory, not before the destination airport, which means that we created a new path starting from the mutation point. It is possible that this process does not successfully finish because we navigated into a dead end and we do not reach the original trajectory. If we cannot join the original trajectory, we repeat this process for several tries until we stop without having this individual mutated.

The process of mutation was, up to this point, only considering changing the path, but the altitude was neglected. Because we also need a target altitude for every segment that we are going to pick the allowed cruising altitude of the segment closest to the target

altitude of the previous segment. This way the aircraft will cruise as steady as possible between the mutation point until it joins the original trajectory.

The entire process of mutation is shown in Algorithm 5.6. We are using the previously defined algorithms for finding a point where the mutation will start. Step by step we choose a random arc while respecting the defined constraints. In this process, we want to avoid using arcs that are definitely closed by restrictions of the FPP instance. Therefore we are already operating on the reduced graph where certain restrictions are already applied. Example 5.4 shows the mutation of an individual shown in Figure 5.12.

Algorithm 5.6: Mutation process

Input: Graph $G = (V, A)$, Individual T, Steps s
Output: Mutated individual

```
1 mp ← mutation_point(G,T)     // randomly pick suitable point
2 p_0 ← sub(T,mp)                          // Subpath up to mp
3 i ← 1
4 while i ≤ s do
5 |   p_i ← p_{i-1} ⊕ random_arc(p_{i-1})
6 |   i ← i + 1
7 end
8 mutated ← join(p_s, T)
9 return p_0
```

Example 5.4 *Consider Figure 5.12 for the following example of a mutation precedure. Let Pt_2 bet the node that was choosen to be the mutation point. The possible outgoing connections of Pt_2 are to Pt_{21} and Pt_{22}, Pt_3 is excluded because it is part of the original individual. The orange dashed line shows the direct connection from the departure airport Dep_{Apt} to the destination airport Dst_{Apt} via the mutation point Pt_2. This distance is the diversion value for the probability calculation for each arc, $diversion_{max}$ will be 115% of that. The individual value for the probability calculation of each arc will be the distance in green up to Pt_1 plus the distance of the arc at hand, plus the distance from the head of the arc to the destination. The distance for choosing the arc (P_2, P_{21}) is marked in green and the distance for the arc (P_2, P_{22}) in yellow.*

Compared to the lateral mutation, the changes to the target altitude are relatively simple. Since the initial climb and final descend are always formed as steep as possible we are not going to make any changes during these phases. However, in between the TOC and TOD, we are going to apply random changes to the target altitude. Additionally, improving the vertical profile will be backed by a local search explained in the next section.

When implementing the described approach, because this is a very practically oriented problem setting, there are many small details and problems that we encountered but have been left out when writing. To go into every detail would exceed the scope of this work.

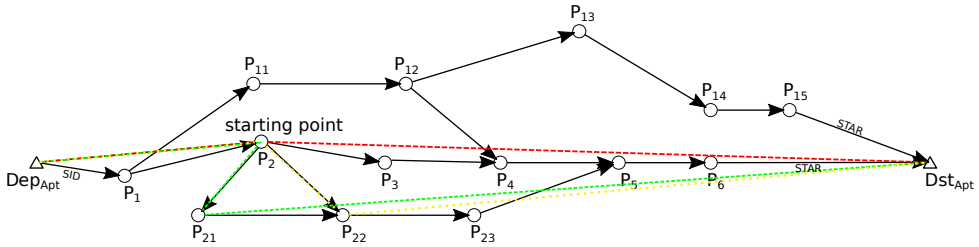


Figure 5.12: Example mutation process at a waypoint P_2. The $diversion_{min}$ is indicated with an orange dashed line. The *diversion* distances for the segment (P_2, P_{21}) and (P_2, P_{22}) are drawn in green and yellow respectively.

Instead, the focus was to capture the essential components and the general functionality that is behind this approach.

5.8 Local search

To improve the vertical profile, we are performing a simple hill-climbing search based on a *local search probability*. In Algorithm 5.7 we can see a local search, first improvement algorithm. This means that the algorithm tries to iteratively improve the current solution by applying a single change, moving towards a local optimum. For all the segments of a solution, we try to increase and decrease the target altitude to the next allowed cruising altitude. A single change at a time is applied, looking for the best possible change. The algorithms can be either implemented with best improvement or a first improvement strategy.

In the former one, all possible changes are evaluated to find the change with the single most improvement to the solution. The latter one stops its search as soon as a move was found that improves the current solution. If an improving move was found, it will be applied, and the process repeats. Once no improving move was found, the algorithm terminates. Additionally, we first try to improve the altitudes starting at the destination because, apart from restrictions, we expect there to be more improvements and want to reduce the computational efforts.

Algorithm 5.7: Local search

```
   Input: Solution S
   Output: Improved trajectory
 1 moves ← [lower(), higher() ]
   /* takes a solution and a segment and increases/decreses
      the cruising altitude to the next altitude according to
      its cruise table                                        */
 2 for move ∈ moves do
 3 |   for (u, v) ∈ reverse(S) do
 4 |   |   if fitness(move(S,(u,v))) < fitness(S) then
 5 |   |   |   S ← move(S, (u,v))
 6 |   |   |   return local_search(S)      // Recursively try again
 7 |   |   end
 8 |   end
 9 end
10 return S
```

CHAPTER 6

Computational results

In this chapter, we first provide information on the data that was used. Next, we present the computational results comparing the reference method from Chapter 4 with the proposed approach from Chapter 5.

6.1 Datasets

All the datasets for navigation data, restrictions, weather, aircraft performance, as well as the APM itself where provided by an industrial partner.

6.1.1 Navigation database

The navigation database consists of 112.023 en route waypoints as part of the global network which are connected by 1.674.990 segments. A total of 599.056 terminal procedure nodes are used by terminal procedures to connect the airports to the global network. At the time the tests were carried out, 20.372 restrictions were active.

Search-space reduction

To reduce the search-space, we are limiting the feasible region for each flight. Only waypoints for which Equation 6.1 holds is included for the search. This reduction is going to inrease the performance of the reference implementation while for the GA approach we do not expect a big impact.

$$dist(Dep_{Apt}, v) + dist(v, Dst_{Apt}) \leq dist(Dep_{Apt}, Dst_{Apt}) \cdot 1.2 \qquad \text{for all } v \in V \quad (6.1)$$

Additionally, the outgoing arcs of each node in the Dijkstra main iteration have been limited to include only segments where the turn angle is less than 170° (ultimately

excluding 20°). In practice such a close angle would not be flown, this again limits the number of considered arcs.

6.1.2 Aircraft performance module

Although this approach is aircraft independent, we choose to use the Airbus A-320 for all our test. The database used for this aircraft contained 1.2034 records for climb, 26.1964 records for cruise and 13.648 descend records. This is a very common aircraft and therefore a good representative of a real-world scenario.

6.1.3 Weather

A single weather set consists of GRIB files with a total size of approximately 850MB. The resolution is 1°, for 16 different altitude levels. For every three hours is a new weather set available. This input data is pre-processed using linear interpolation to get weather information on every 1000 feet of altitude in 1-hour increments. The final weather data used in the optimization uses around 1GB of memory.

The weather module retrieves the closest available data point for any requested location, altitude, time tuple. Because of the already high resolution, computationally expensive interpolation can be saved while having very accurate results.

SIGMETs are integrated in the form of restricted areas. This allows to avoid areas with dangerous weather phenomena without any additionl effort.

6.1.4 GA Computation Improvements

To increase computation speed, a cache for calculating the penalty of a trajectory caused by restrictions was used. This cache contained the path of every lateral route that was certainly not valid, independent from the vertical profile. Before calculating the penalty of a trajectory it is always first checked if it is a known invalid path.

6.2 Test Environment

Both, the approach based on Dijkstras algorithm as well as our genetic approach are implented in C++14 and compiled with Clang 5.0.1-4 using full compiler optimization. Our computations were performed on machines with an Intel® Core™i7-7700 processor with 3.60GHz and 32GB of RAM running Ubuntu 18.04.1 LTS.

6.3 Instances

In Table 6.1 we can see a list of the busiest airports in Europe from the year 2016. The last column shows the ICAO code for identification in the test-cases. For our tests we picked connections from this list, each city used in a test-instance is written in bold, for

#	Country	Airport	City	ICAO
1	United Kingdom	Heathrow	London	**EGLL**
2	France	Charles de Gaulle	Paris	**LFPG**
3	Netherlands	Amsterdam Schiphol	Amsterdam	EHAM
4	Germany	Frankfurt	Frankfurt	EDDF
5	Turkey	Istanbul Atatürk	Istanbul	LTBA
6	Spain	Adolfo Suárez Madrid–Barajas	Madrid	LEMD
7	Spain	Barcelona El Prat	Barcelona	LEBL
8	United	Kingdom London-Gatwick	London	EGKK
9	Germany	Munich	Munich	**EDDM**
10	Italy	Leonardo da Vinci–Fiumicino	Rome	**LIRF**
11	Russia	Sheremetyevo International	Moscow	UUEE
12	France	Paris-Orly	Paris	LFPO
13	Turkey	Sabiha Gökçen	Istanbul	LTFJ
14	Denmark	Copenhagen	Copenhagen	**EKCH**
15	Russia	Domodedovo International	Moscow	UUDD
16	Ireland	Dublin	Dublin	**EIDW**
17	Switzerland	Zürich	Zürich	LSZH
18	Spain	Palma de Mallorca	Palma de Mallorca	LEPA
19	Norway	Oslo Gardermoen	Oslo	**ENGM**
20	United Kingdom	Manchester	Manchester	**EGCC**
21	Sweden	Stockholm-Arlanda	Stockholm	ESSA
22	United Kingdom	London Stansted	London	EGSS
23	Germany	Düsseldorf	Düsseldorf	EDDL
24	Austria	Vienna International	Vienna	**LOWW**
25	Portugal	Lisbon Portela	Lisbon	LPPT

Table 6.1: Busiest airports in Europe 2016, source Wikipedia

easier reference. However, we did not include two airports from the same country, instead tried to spread out the instances.

In Table 6.2 we can see the result of a single run of an optimization from EDDF to ENGM. The first three columns show the trip cost and objective value (which are omitted for the valid routes because they are equal) of the best valid and invalid individual. The initial population consists of 23 individuals, which is the number of pre-existing routes that were available for this city pair. The column "population size" is the absolute number of individuals that we started with. Also contained in the initial population is the result (including intermediate invalid results) of a previously run optimization using the exact algorithm without any wind. Integrating this pre-calculated route improved the convergence rate significantly.

The termination criterion was configured to be five successive iterations without any

improvement. This limit seems low, but preliminary tests showed that no improvement was made by increasing the limit, as long as the results of a "no wind" run were included in the initial population. A population size of 75 was used, while the best invalid and invalid individual always were carried to the subsequent population (which explains the population size of 76 and 77). Increasing the population size has a big impact on the computation time, largely because of the time-consuming evaluation of restrictions.

Table 6.3 shows the results of all performed tests on the 20^{th} of January 2019. For every city pair we tested, we performed 50 runs of the heuristic approach. For the optimizations using the exact approach, we added a maximum re-optimization limit of 50, after which the best valid solution (if one found) was returned without having all queued re-optimizations handled. All tests used the aircraft A320-232 with 75 tons takeoff mass. A population size of 75 individuals was used. The termination criterion was configured to stop after 5 consecutive iterations without an improvement. The initial population consisted of preferred routes as well as the final and intermediate results of a previously performed optimization using the exact method under no-wind conditions.

	Valid	Invalid		population size			Paths		
	trip cost	objective	trip cost	popsize	valid	discovered	valid	infeasible	Time
0		∞	5481.93	23	0	0	16	3	1.164421s
1		∞	5867.96	76	0	33	35	22	3.578540s
2		∞	5867.96	76	0	53	55	42	5.683697s
3		∞	5867.96	76	0	62	64	50	7.444512s
4		∞	5867.96	76	0	71	73	59	9.545499s
5		∞	5840.1	76	0	83	85	70	11.900863s
6		∞	5840.1	76	0	93	95	80	14.624752s
7		∞	5840.1	76	0	98	100	85	17.475951s
8	6271.74	6271.74	6271.74	76	2	108	110	93	20.136539s
9	6271.74	6271.74	6271.74	77	9	120	122	104	23.960210s
10	6271.74	6271.74	6271.74	77	36	128	130	109	30.368625s
11	6271.74	6271.74	6271.74	77	48	140	142	116	33.892488s
12	5976.38	5976.38	5976.38	77	54	152	154	125	40.351502s
13	5976.38	5976.38	5976.38	77	56	160	162	131	47.881471s
14	5976.38	5976.38	5976.38	77	48	169	171	138	52.444768s
15	5976.38	5976.38	5976.38	77	50	176	178	143	62.347211s
16	5976.38	5976.38	5976.38	77	57	185	187	149	66.527564s
17	5976.38	5976.38	5976.38	77	52	193	195	157	72.132413s

Table 6.2: Result of an optimization from `EDDF` to `ENGM`

		Reference system			Proposed heuristic approach							
					Cost				Time			
Dep	Dst	Cost	Time	Re-opt.	Avg	Median	Min	Max	Avg	Median	Min	Max
LEPA	EDDL	7157.10	360	39	7171.39	7171.53	7170.28	7171.92	147.08	139	75	244
LOWW	EGLL	6051.13	349	50	6176.91	6175.48	6175.48	6187.01	76.68	76.5	36	117
EGCC	LOWW	6763.97	49	3	6755.26	6755.18	6755.18	6757.07	118.86	115	71	191
EDDF	LOWW	3684.56	60	8	3675.64	3675.64	3675.64	3675.64	40.26	39	28	59
LOWW	ESSA	6152.76	20	1	6141.00	6141.29	6139.97	6141.58	11.14	11	10	13
LOWW	LFPG	5154.90	77	2	5285.36	5285.51	5285.06	5285.51	68.62	67	54	102
ENGM	EDDF	5470.09	15	4	5312.66	5311.29	5311.29	5336.43	73.32	71.5	47	131
LOWW	EDDL	4479.23	362	50	4318.07	4318.07	4318.07	4318.07	74.72	71	57	115
EGLL	ESSA	6814.38	81	8	6795.98	6795.98	6795.98	6795.98	19.48	19	16	27
EGLL	LOWW	6064.52	75	7	6055.33	6055.33	6055.33	6055.33	47.78	47	42	63
LOWW	ENGM	6738.23	28	1								
EDDL	LEPA	6764.54	347	50	6723.46	6722.78	6722.76	6725.98	108.2	103	62	173
EDDF	ENGM	5705.94	105	10	5991.58	5972.26	5882.40	6273.00	61.78	61	30	98
ESSA	EIDW	7541.32	14	1	7590.05	7594.01	7554.66	7660.06	15.4	15	12	22
EKCH	LIRF	6934.85	428	50	6825.19	6824.31	6816.04	6846.57	133.74	130.5	89	223
EGLL	ENGM	5971.36	19	1	5973.52	5973.52	5973.52	5973.52	18.38	18	15	23
EKCH	EDDM	4305.50	39	2								
ESSA	EGLL	6886.23	51	3	6886.20	6886.20	6886.20	6886.20	19.02	18.5	15	26
ESSA	EGCC	7276.72	370	47	7333.32	7333.44	7330.09	7333.44	38.26	37.5	31	63
EGCC	EKCH	5184.33	143	16	5185.67	5185.99	5185.01	5185.99	15.98	16	12	22

Table 6.3: Results from 20 january 2019

Figure 6.1 shows the cost difference of the result of the exact algorithm as well as the average and median costs of our result.

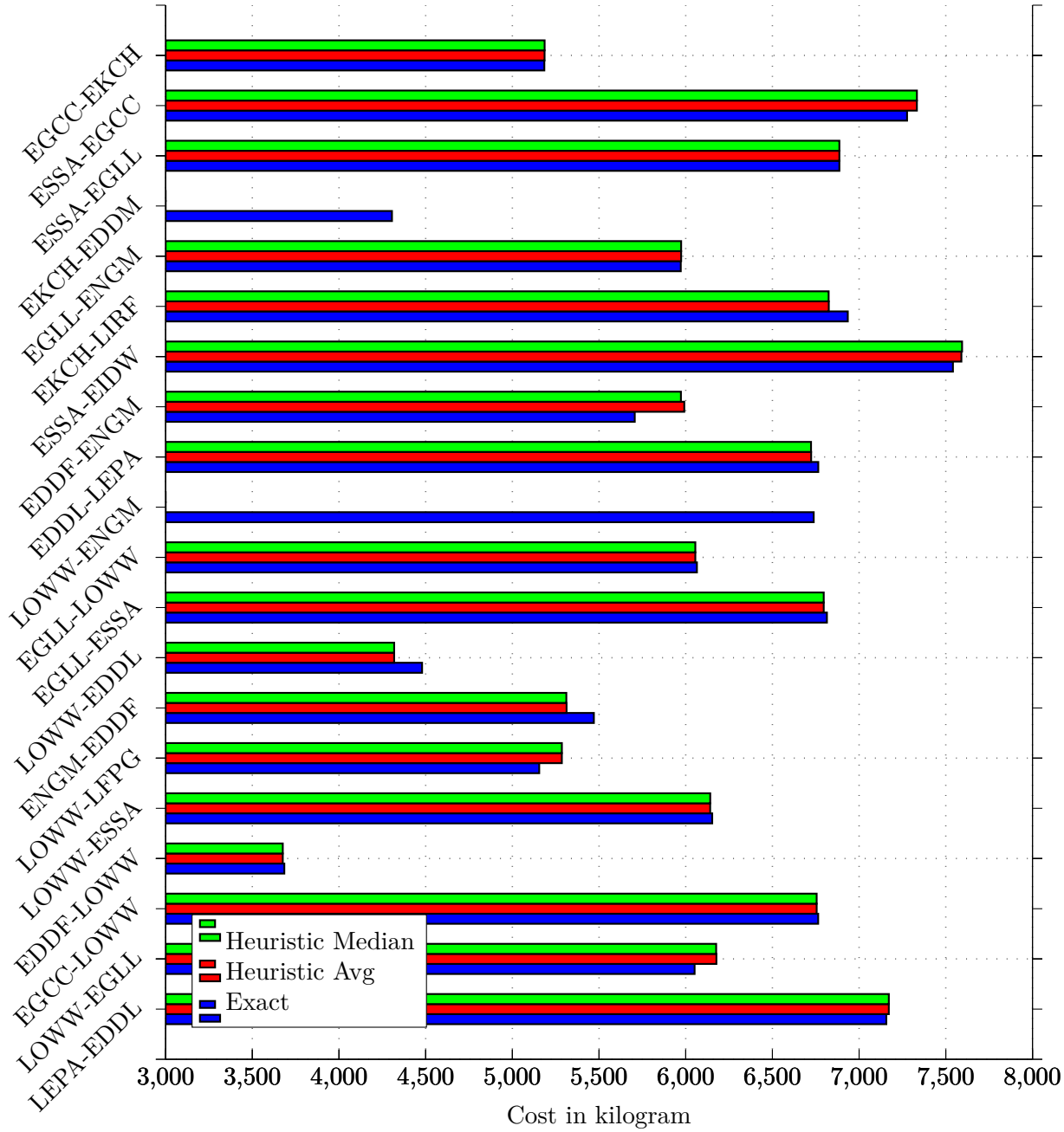


Figure 6.1: Cost comparison

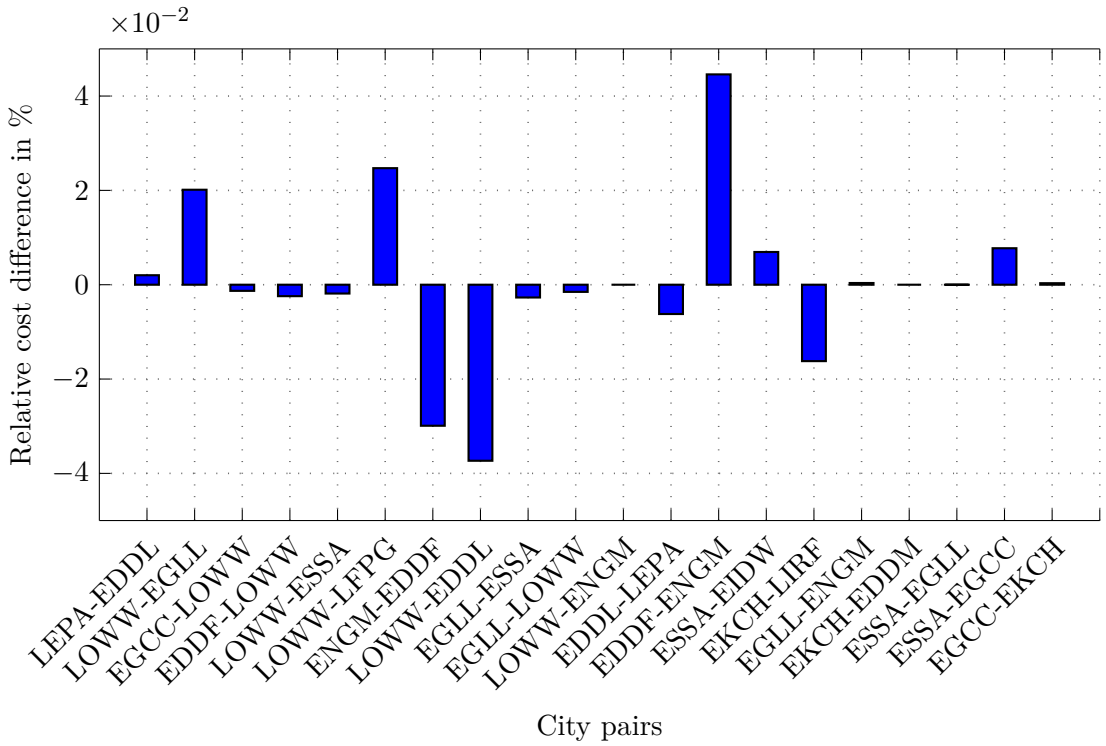


Figure 6.2: Relative cost differences

To illustrate the difference, Figure 6.2 shows the percentual difference of the exact method vs. the median of our approach. It can clearly be seen that the maximum difference is less than 0.05 percent. We compared the resulting trajectories that turned out the be exactly the same and we blame this small offset on the different implementations.

Figure 6.3 visualizes the different time spent for both approaches. While returning the same results as the exact method, our approach reached much higher performance in essentially all cases. Also worth noting is that for the reference approach An interesting observation of the reference system can be made when looking at instances for which both directions have been calculated. Although there is are city pairs for which the computation time for both directions is similar, like `ESSA` to `EGLL`, 50 vs. 80 seconds and `EDDL` to `LEPA` with more than 340 vs. 360 seconds there we also have `EDDF` to `ENGM` with 105 vs. less than 20 seconds and finally `EGLL` to `LOWW` with 80 vs. 350 seconds. This significant difference in computation time is inflicted by restrictions causing re-optimizations, confirming that the reference approach has difficulties coping with the restrictions created by ATC. On the other hand, an impact to such an extent is not observable to our approach.

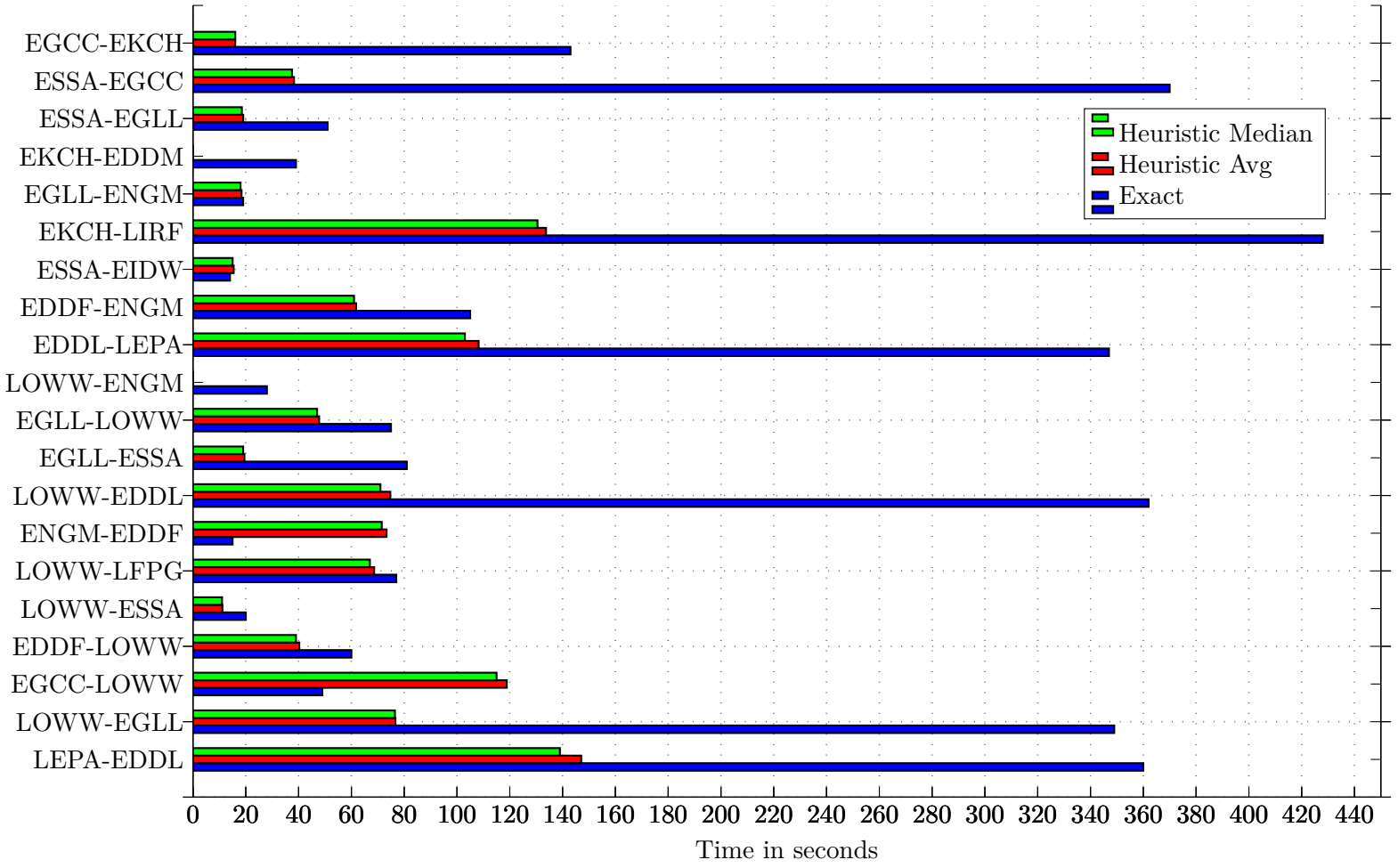


Figure 6.3: Time comparison

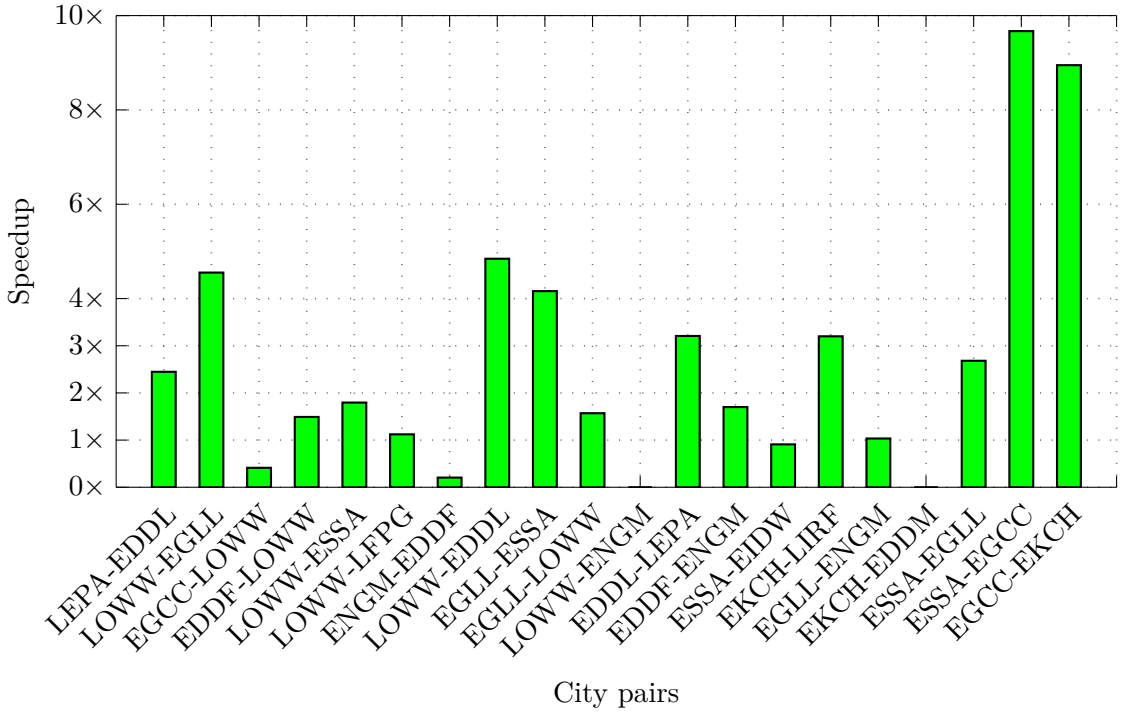


Figure 6.4: Speedup by our heuristic approach

Figure 6.4 shows the speedup in computation time reached by our heuristic (average of 50 runs) compared to the reference method.

CHAPTER 7

Conclusion

In this final concluding chapter we sumarize the contributions of this work and point thowars possible directions for future work.

7.1 Concluding remarks

We successfully applied a new heuristic approach to the FPP. Although there are two instances for which the exact method outperformed our approach by half, the average and maximum time spent on optimization are significantly lower with our approach. As the computational experiments showed, besides the two instances in which our approach did not manage to find a valid solution, our method managed to return the same result as found by the exact approach. Reasons for that might be the comparably low number of initial solutions we started the algorithm, which might be overcome by randomly generated initial population.

Because in our approach it is possible to integrate the result of a previous computation, we do not need to perform a complete optimization from the beginning, but instead, have a hot-start continuing using the previous results. This hot-start also makes it possible to consider several iterations with different take-off masses very quickly to figure out how much fuel needs to be tanked. Furthermore, our approach allows for an easier adaption to external changes like updated weather information, better estimate about the aircraft's payload.

7.2 Summary of thesis contributions

Among other things the main contributions of this work are:

- In this work, we specified the FPP as it is encountered in practice when planning a trajectory.

- A reference implementation, based on a exact method, like it is used in many solutions in the industry, has been explained and implemented.
- Next, a new heuristic method has been proposed for solving the problem. Different variations for solution representation have been discussed.
- A method was shown that allows precluding if for a given lateral path there exists a vertical profile that does not violate any restrictions
- Both versions have been evaluated in real-world scenarios and results were collected. The results were analyzed comparing the two presented approaches.

7.3 Future Work

Many interesting tests and evaluations have been left unanswered because of the lack of time. The question that is most likely to be asked is how well does the algorithm adapt to changes in the network structure, restrictions, and SIGMETs. The algorithm has primarily taken Eurocontrols preferred routes as a strating point, however it would be interesting to see the performance using an initial population made out of solutions generated by procedure similar to our mutation process.

One of the initial goals set for this work was to find a solving method that allows being extended to extend the problem further. The reason for this goal was that there is already a need to further extend the scope of the optimization process by including additional variables. The following list is only an abstract of interesting directions to go from here.

- As already mentioned, there was one cost component left out in this work which is cost induced by overflight charges. These can be implemented by just adding the necessary formulas and adding the resulting cost to the total cost of the trajectory.
- Instead of using only one weather model for the optimization it is possible to consider multiple different weather predictions. These so-called weather ensembles are a set of different weather forecasts, based on different initial parameters. Depending on the desired outcome, one could evaluate a single trajectory for all weather forecasts in the ensemble and generate a combined objective value.
- Extended Range Twin Operations (ETOPS) is a relaxation of the safety measures which allows carrying less additional fuel for unforeseen emergencies than usual as long as some other safety regulations are respected. One of those regulations is that a maximum duration in which an alternate airport can be reached, in case of an engine failure, must not be exceeded at any point along the trajectory. This can be integrated as a penalty to the objective function in trajectories where this condition was not respected.

- To further improve the performance of this approach, it can naturally make use of multithreading. An implementation for a distributed environment is also possible with our method.

List of Figures

List of Tables

List of Algorithms

Acronyms

AIP Aeronautical Information Publication. 20

AIRAC Aeronautical Information Regulation And Control. 20

APM Aircraft Performance Model. 22, 23, 28, 34, 37, 65

ATC Air Traffic Control. 1, 3, 5, 8, 13, 14, 17, 31, 33, 39, 43, 47, 54, 56, 58, 72

CI Cost Index. 26

ESAD Equivalent Still Air Distance. 26

ETOPS Extended Range Twin Operations. 76

FMS Flight Management System. 26, 29

FRA Free route airspace. 3, 12

GRIB GRIdded Binary. 21, 66

ICAO International Civil Aviation Organization. 8, 11, 66

ISA International Standard Atmosphere. 20, 21, 34

RAD Route Availability Document. 20

RoC Rate of Climb. 22, 23

RoD Rate of Descent. 22, 23

RVSM Reduced Vertical Separation Minimum. 14

SID Standard Instrument Departure. 11

SIGMET Significant Meteorological Information. 21, 22, 66, 76

STAR Standard Terminal Arrival Route. 11–13, 79, 81

TAS True Air Speed. 23

TOC Top of climb. 7, 47, 61

TOD Top of descend. 8, 47, 53, 54, 61, 79

WGS84 World Geodetic System 1988. 24

Bibliography

[1] Eurocontrol. Challenges of growth, 2013. [Online; access 5-Dec 2018] http://www.eurocontrol.int/articles/challenges-growth.

[2] Eurocontrol. Long-term forcast flight movements 2010-2030. [Online; accessed 5-Dec 2018] "http://www.eurocontrol.int/sites/default/files/publication/files/long-term-forecast-2010-2030.pdf".

[3] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[4] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[5] Wikipedia. Optimal control. [Online; accessed 5-Dec 2018] "https://en.wikipedia.org/wiki/Optimal_control".

[6] Antonio Franco and Damián Rivas. Optimization of multiphase aircraft trajectories using hybrid optimal control. *Journal of Guidance, Control, and Dynamics*, 38(3):452–467, 2015.

[7] Alfonso Valenzuela and Damián Rivas. Analysis of along-track variable wind effects on optimal aircraft trajectory generation. *Journal of Guidance, Control, and Dynamics*, 39(9):2149–2156, 2016.

[8] Z. and É. Dynamic programming for vertical flight planning. 2016.

[9] Christian Kiss-Toth and Gabor Takacs. A dynamic programming approach for 4D flight route optimization. In *2014 IEEE International Conference on Big Data (Big Data)*, pages 24–28. IEEE, 2014.

[10] Anders N. Knudsen, Marco Chiarandini, and Kim S. Larsen. Heuristic variants of A* search for 3D flight planning. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 361–376. Springer International Publishing, 2018.

[11] Alejandro Murrieta Mendoza and Ruxandra Botez. Vertical navigation trajectory optimization algorithm for a commercial aircraft. In *AIAA/3AF Aircraft Noise and Emissions Reduction Symposium*. American Institute of Aeronautics and Astronautics, 6 2014.

[12] Roberto Salvador Félix Patrón and Ruxandra Mihaela Botez. Flight trajectory optimization through genetic algorithms for lateral and vertical integrated navigation. *Journal of Aerospace Information Systems*, 12(8):533–544, 8 2015.

[13] Roberto S Félix Patrón, Yolène Berrou, and Ruxandra M Botez. New methods of optimization of the flight profiles for performance database-modeled aircraft. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 229(10):1853–1867, 2014.

[14] SESAR. Sesar joint undertaking | vision. [Online; accessed 23-Feb 2019] "http://www.sesarju.eu/vision".

[15] International Civil Aviation Organisation. Phase of flight, definitions and usage notes (1.3), April 2013. [Online; accessed 5-Dec 2018]"http://www.intlaviationstandards.org/Documents/PhaseofFlightDefinitions.pdf".

[16] Eurocontrol. Free route airspace. [Online; accessed 5-Dec 2018] "http://www.eurocontrol.int/articles/free-route-airspace".

[17] ICAO - appendix g to part 91—operations in reduced vertical separation minimum (RVSM) airspace. https://www.icao.int/safety/fsix/Library/AppendixG_to_Part-91.pdf[Online; accessed 03-Dec 2018].

[18] Anders Nicolai Knudsen, Marco Chiarandini, and Kim S. Larsen. Constraint handling in flight planning. In J. Christopher Beck, editor, *Principles and Practice of Constraint Programming*, volume 10416 of *Lecture Notes in Computer Science*, pages 354–369, Cham, 2017. Springer International Publishing.

[19] Route availability document. http://www.nm.eurocontrol.int/RAD/ [Online; access 5-Dec 2018].

[20] World Meteorological Organization. *Manual on Codes*. Secretariat of the World Meteorological Organization, 2010.

[21] Alejandro Murrieta-Mendoza, Laurane Ternisien, Bruce Beuze, and Ruxandra Mihaela Botez. Aircraft vertical route optimization by beam search and initial search space reduction. *Journal of Aerospace Information Systems*, 15(3):157–171, mar 2018.

[22] Alejandro Murrieta-Mendoza, Antoine Hamy, and Ruxandra Mihaela Botez. Four- and three-dimensional aircraft reference trajectory optimization inspired by ant

colony optimization. *Journal of Aerospace Information Systems*, 14(11):597–616, nov 2017.

[23] Eurocontrol. Base of aircraft data. [Online; accessed 5-Dec 2018] "http://www.eurocontrol.int/services/bada".

[24] Wikipedia. Illustration of great-circle distance. [Online; accessed 5-Dec 2018] "https://en.wikipedia.org/wiki/Great-circle_distance".

[25] H.N. Gabow, S.N. Maheshwari, and L.J. Osterweil. On two problems in the generation of program test paths. *IEEE Transactions on Software Engineering*, SE-2(3):227–231, 1976.

[26] Melanie Mitchell. Handbook of genetic algorithms (l. d. davis). *Artif. Intell.*, 100:325–330, 1998.

[27] Chang Wook Ahn and R.S. Ramakrishna. A genetic algorithm for shortest path routing problem and the sizing of populations. *IEEE Transactions on Evolutionary Computation*, 6(6):566–579, 2002.

[28] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.