Departure and arrival routes optimisation approaching big airports

Jérémie Chevalier

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Université Toulouse 3 - Paul Sabatier

Présentée et soutenue par

Jérémie CHEVALIER

Le 28 septembre 2020

Optimisation des routes de départ et d'arrivée aux approches des grands aéroports

---

Ecole doctorale : **AA - Aéronautique, Astronautique**

Spécialité : **Mathématiques et Applications**

Unité de recherche :

Thèse dirigée par

Pierre MARECHAL et Mohammed SBIHI

Jury

M. Xavier PRATS, Rapporteur
M. Adnan YASSINE, Rapporteur
M. Daniel DELAHAYE, Examinateur
Mme Fulya AYBEK ÇETEK, Examinatrice
M. Valentin POLISHCHUK, Examinateur
M. Pierre MARECHAL, Directeur de thèse

Résumé

L'objectif de cette thèse est de proposer une méthodologie pour la conception optimale des routes de départ et d'arrivée dans les espaces aériens entourant les grands aéroports, appelés Terminal Maneuvering Areas (TMA). Les routes que suivent les aéronefs pour décoller et atterrir sont respectivement appelées Standard Instrument Departures (SID) et Standard Terminal Arrival Routes (STAR), et sont aujourd'hui créées à la main par des experts métier. Cependant, l'augmentation prévue du trafic aérien à l'échelle mondiale va causer de plus en plus de problèmes de congestion dans les années à venir. Pour contrer ces problèmes, les différents acteurs du trafic aérien prennent des mesures, pour des améliorations à court et à long terme. L'une de ces mesures consiste à faire une meilleure utilisation de l'espace aérien, à commencer par les TMA, qui sont les plus engorgées car elles représentent un espace restreint contenant les points de départ et d'arrivée du trafic. Cette réorganisation de l'espace aérien, qui passe notamment par une meilleure construction des SIDs et STARs, ou de leur amélioration lorsqu'elles existent déjà, doit non seulement répondre de manière pertinente à la situation de congestion croissante mais doit aussi prendre en compte diverses contraintes opérationnelles et enjeux environnementaux. Ainsi nous nous intéressons dans cette thèse au problème de conception optimale de ces routes aériennes. Cette optimisation est effectuée en prenant en compte les contraintes suivantes : navigablilité des routes (c'est à dire la facilité qu'un appareil donné aura à la suivre, l'inconfort occésionné pour les passagers etc.), évitement des obstacles, séparation des routes entre elles, séparation des points de connexion des routes. Les points de connexion représentent l'endroit où deux routes provenant de points d'entrée différents de la TMA se rejoignent pour n'en former qu'une, ou au contraire l'endroit où une route est séparée en deux pour atteindre deux points de sortie différents de la TMA. Un meilleur placement de ces points de connexion permet de réduire la charge mentale des contrôleurs dans la gestion du trafic. L'objectif quant à lui consiste à réduire la longueur des routes, l'espace total qu'elles occupent, ainsi que le bruit subi par les populations survolées par celles-ci.
Nous commençons par formuler le problème comme un problème d'optimisation mathématique. Dans ce modèle, les routes sont modélisées en 3D et sont composées de deux parties. La première partie de la route, représentant sa trace horizontale, est une succession de nœuds dans un graphe discrétisant la base de la TMA. La seconde partie est un cône d'altitudes représentant pour chaque point de la partie horizontale l'intervalle d'altitudes dans lequel un aéronef peut se trouver à ce point de

la route. Les obstacles sont représentés par des cylindres à base polygonale, et les villes par des polygones en 2D auxquels sont associées des fonctions indiquant la densité de population en chaque point de la surface polygonale. Les fonctions densité permettent de définir le coût du bruit.
Pour la résolution du problème, deux approches sont proposées. La première est une heuristique basée sur le principe de la programmation dynamique. Nous montrons d'abord que le sous-problème de la conception d'une seule route (SID ou STAR) peut se ramener à un problème du plus court chemin contraint dans un graphe, réputé NP-difficile. Pour des tels problèmes, le principe de programmation dynamique peut être formulé, avec comme variables d'état des étiquettes associées aux nœuds du graphe, mais contrairement au problème classique du plus court chemin il faut conserver un ensemble d'étiquettes à chaque nœud, ce qui mène à une complexité de résolution exponentielle. Pour y remédier, le principe est utilisé de façon heuristique : seul le plus court chemin jusqu'à un nœud donné est retenu, ce qui revient à se placer dans le cas d'une recherche sans contrainte. Ces contraintes son prises en compte par une phase de pré-traitement, qui associe à chaque nœud un intervalle d'altitudes autorisées. Lors de la recherche de chemin, un chemin peut passer par un nœud donné uniquement si l'altitude atteinte à ce nœud se trouve à l'intérieur de l'intervalle calculé. Cette heuristique est ensuite étendue à la conception de plusieurs routes. En plus de la sous-optimalité induite par l'utilisation heuristique de la programmation dynamique, cette heuristique suppose un ordre *a priori* pour la génération des routes ce qui peut dégrader davantage la qualité de la solution obtenue. Nous proposons donc une deuxième approche.
La seconde approche est basée sur l'utilisation d'une métaheuristique : le recuit simulé (SA pour Simulated Annealing). Dans cette approche, une solution est construite route par route puis optimisée pour le critère donné. Le fait de construire les routes dans un ordre donné n'est pas un problème ici, car une composante aléatoire est introduite dans la recherche de route. La méthode repose sur deux étapes majeures : génération d'une solution, et production d'une solution voisine. Dans la génération d'une solution, chaque route est construite à l'aide d'une recherche de chemin dans le graphe discrétisant la base de la TMA. Le coût des arcs du graphe est préalablement défini avec une méthode spécifique, permettant de donner n'importe quelle forme aux routes en 2D. Par ailleurs, la partie verticale des routes est calculée sur la base de pentes minimale et maximale, auxquelles peuvent s'ajouter des vols en palier, pour répondre à des contraintes opérationnelles ou pour franchir des obstacles. Certaines contraintes, telles que l'évitement des obstacles, sont relâchées et intégrées à la fonction objectif. Les routes sont générées par ordre décroissant d'importance des flux. Afin de gérer leurs points de fusion, le processus fonctionne comme suit : la première route est générée, de la piste jusqu'à son point d'arrivée. Ensuite, la deuxième route se connecte sur la première en un point défini par le SA. La troisième route se connecte sur une des deux précédentes, en un autre point bien défini et ainsi de suite. Une fois cette solution générée, une solution voisine est construite en altérant la solution initiale, par exemple en modifiant la forme d'une route, en ajoutant ou supprimant un vol en palier, ou en déplaçant un point

de connexion. L'une des deux solutions est ensuite choisie comme nouvelle solution initiale, et les étapes précédentes sont effectuées à nouveau, jusqu'à vérifier une condition d'arrêt.
Les deux approches ont d'abord été testées sur un cas test artificiel construit de façon ad hoc pour permettre de mesurer leurs forces et faiblesses ainsi que leur sensibilité aux paramètres de modélisation et algorithmiques. Ces tests sont réalisés de façon incrémentale : sans ou avec un, ou plusieurs obstacles ; une ou plusieurs routes. Ces tests ont permis de mettre en lumière certaines limites de la première approche, basée sur le principe de programmation dynamique, que la méthode basée sur le recuit simulé ne présente pas, ou dans une forme très atténuée. De ce fait, seule cette dernière approche a été confrontée aux autres cas tests. Ces cas correspondent à des TMA réelles, tirées de la littérature : Stockholm, Paris Charles-de-Gaulle et Zurich. Chacun de ces tests a permis de mieux mesurer et comprendre les performances de notre algorithme. L'exemple de Stockholm montre l'importance du choix dans la manière de construire le graphe pour le résultat final. L'exemple de Charles-de-Gaulle nous a permis de tester les limites de notre méthode dans un cas comportant un grand nombre de routes à construire, réparties sur quatre pistes. L'exemple de Zurich montre les limites de l'algorithme dans un cas comportant un grand nombre d'obstacles. En dépit de ces limites, les tests effectués montrent que les performances de notre algorithme sont satisfaisantes, comparé à l'état de l'art ainsi qu'aux opérations en place actuellement, bien que des améliorations puissent être apportées. Notre méthode doit être prise comme un outil d'aide à la décision pour les concepteurs de procédures, capable de fournir rapidement une première solution dans les cas de conception de routes complexes.

Abstract

This thesis aims at proposing a way to automatically design optimal departure and arrival routes in the airspace surrounding large airports, called Terminal Maneuvering Area (TMA). The routes that aircraft follow to depart from and arrive to airports are respectively called Standard Instrument Departure (SID) and Standard Terminal Arrival Routes (STAR), and are currently designed by hand by experts. However, the predicted increase in air traffic worldwide is bound to cause more and more congestion issues in the years to come. In order to address that issue, various measures are taken by the different actors of air traffic for short to long term improvements. One of these measures is to make a more efficient use of the airspace, beginning with TMAs, which are the most congested, as they represent a narrow space containing the departure and arrival points for the traffic. This reorganization of the airspace, that involves a better construction of the SIDs and STARs, or their improvement when they already exist, must not only provide an adequate solution to the situation of increasing congestion, but it must also take into account various operational constraints and environmental concerns. In this thesis, we address the problem of the optimal design of these air routes. This optimization is carried out by taking into account the following constraints: "flyablility" of the route (which measures how easy it is for a given aircraft to follow the route, the discomfort for the passengers etc.), obstacle avoidance, pairwise route separation, separation of the merging points of the routes. The merging points represent the location where two routes coming from different entry points of the TMA merge into one, or, conversely, where a route splits into two routes in order to attain two different exit points of the TMA. A better placing of these merging points allows for a lower workload for the controllers in the air traffic management. The objective consists in reducing the length of the routes, the total space they occupy, as well as the noise disturbance induced for the populations flown over.
We begin by formulating the problem as a mathematical oprimization problem. In this model, the routes are considered in 3D and are represented in two parts. The first part of a route, which represents its horizontal component, is a sequence of nodes in a graph structure obtained by sampling the TMA. The second part is a cone of altitudes representing the range of altitudes for each point along the horizontal part of the route that an aircraft is able to attain at this point following the route. The obstacles are modeled as cylinders with polygonal bases, and the cities by 2D polygons associated to a function indicating the population density on each point of the polygon.

The population density functions allow to define the cost relative to noise. For the problem resolution, two approaches are proposed. The first one is a heuristic based on the dynamic programming principle. We first show that the subproblem of the design of one route (SID or STAR) can be assimilated to the problem of finding the shortest constrained path in a graph, which is NP-hard. For this problem, the principle of dynamic programming can be formulated, with the state variables set as labels associated to the vertices of the graph. However, as opposed to the usual, non-constrained shortest path finding case, several labels must be kept for each vertex, leading to an exponential resolution time. In order to solve this issue, the dynamic programming principle is considered heuristically: only the shortest path to each vertex is kept as a label, and the other paths are discarded, as in the non-constrained shortest path finding problem. The constraints are taken into account by means of a preprocessing phase which allocates an interval of authorized altitudes for each vertex. During the path search phase, a path can pass through a given vertex only if the altitude attained at this vertex lies within the range of authorized altitudes. This heuristic is then broadened to allow for the design of several routes. On top of the sub-optimality of the solution provided by this method, induced by the use of the dynamic programming principle by the means of a heuristic, this method requires an *a priori* order for the design of the routes, which can further deteriorate the quality of the obtained solution. Thus, we propose a second approach to solve the problem.

The second approach is based on the use of a metaheuristic: the Simulated Annealing (SA). In this approach, a solution is constructed one route at a time, then it is optimized for the given criterion. Here, designing the routes in a given order is not an issue, as randomness is introduced in the design of the routes. This method relies on two main steps: generating a solution, and producing a neighbor solution. In the solution generation, each route is constructed by the means of a path search in the graph that discretizes the base of the TMA. The costs of the edges in the graph are priory set to carefully chosen values, allowing the possibility to give any desired shape to the route in 2D. Additionally, the vertical part of the route is determined by a minimum and a maximum slope, to which level flights can be added in order to comply with operational constraints or to avoid obstacles. Some of the constraints, such as the obstacle avoidance, are relaxed and integrated into the objective function. The routes are generated by decreasing order of traffic flow. In order to manage their merging together, the process works as follows: the first route is created, going from the runway to its ending point. Then, the second route connects to the first one on a point chosen by the SA. The third route connects on one of the previous two on another chosen point, and so on. Once the solution is generated, a neighboring solution is created, by altering the initial solution, for instance by modifying the shape of a route, by adding or by removing a level flight, or by moving a merging point. One of the solutions is then chosen as the new initial solution, and the previous steps are taken again, until a stopping criterion is met.

The two approaches were first tested on an artificial case built in a way that allows to measure their strengths and weaknesses, as well as their sensitivity

to the modeling and algorithmic parameters. These tests were performed in an incremental way: with or without one, or several obstacles; one or several routes. These tests allowed to shed some light on some of the limitations of the first approach, based on the dynamic programming principle, that don't appear with the method based on the simulated annealing, or in a much lighter form. Therefore, only the latter approach was tested against the subsequent test cases. These cases correspond to real-life TMAs, taken from the literature: Stockholm, Paris Charles-de-Gaulle and Zurich. Each one of these tests helped in measuring and understanding the performances of our algorithm. The Stockholm instance showed the importance of the way to construct the graph in the final result. The Charles-de-Gaulle instance allowed to test the limits of the method in a case where many routes, related to four different runways, were to be designed. The Zurich instance showed the limits of the algorithm in a case containing many obstacles. Despite these limitations, these tests showed that our algorithm's performances are satisfactory, relatively to both state-of-the-art methods and current air traffic operations, although there is room for improvement. Our method should be viewed as a decision-helping tool for expert procedure designers, that is able to provide a quick first solution to a complex route design problem.

Acknowledgments

I would like to thank all the people that helped me, directly or not, in achieving this work. These years have been the opportunity for me to work on a topic and in a context that I love, alongside great people. They allowed me to create an algorithm that I'm proud of, and to travel to countries that I always dreamt of visiting. For that I am most grateful.
Firstly, my thoughts go to my supervisors, Mohammed Sbihi, Pierre Maréchal and Daniel Delahaye. These three great minds and kind human beings guided me through my thesis, with great advice on its scientific, methodological and algorithmic aspects. They have always been helpful and caring, and it has been a genuine pleasure to share these years of my life with them. I look forward to our paths crossing again.
I extend my warmest gratitude to the members of my jury: the reviewers Xavier Prats and Adnan Yassine, and the examiners Valentin Polishchuk and Fulya Aybek Çetek. They took the time to review my work, and expressed interest in it which, coming from such experts in their domain, is a great honor to me. They have been nothing but benevolent, and their insight on my work helped me explore the topic in a broader way.
I also want to thank CGX Aéro for giving me the opportunity to carry out this project. It has been the occasion to meet wonderful and caring people, who are too numerous to be all named here but who will take the credit they are entitled to. Nevertheless, I would like to stress my thanks to a few people in particular: Patrice Gonzalez, for always lending an ear to my requests, suggestions and feelings in the company; Pierre-Maël Mayne, even though he left, for welcoming me and for getting me acquainted with the company and with new algorithms in all his sassiness; and Antoine Charpentier for sharing his insights, and the office with me.
I would also like to thank the other PhD students, with whom I shared this singular experience. It has been a pleasure exchanging points of view on our work, but also blowing off some steam with them. They made this time way more pleasant than it could have been, had I been on my own. My special thanks to Romaric, Florian, Sana, Philippe, Gabriel and Jun, for their very enjoyable company.
My thanks also go to all of the Z building, full of people who are as nice as they are competent. A special thanks to Serge Roux for his time and help in all my connectivity endeavors, especially the day of my defense.
I acknowledge the Région Occitanie, that partly funded my research, and allowed me to travel to present my work abroad.
I thank my friends and family, who have always been supportive, and here

for me whenever I needed to. No one could dream of better company than theirs.

To my son, James, who arrived just in time to see his father become a doctor, and last on this page but first in my heart, to my wife, Nathalie, who stood by me even in the darkest moments. Words will never be enough to express how much I love you. Thank you for being the extraordinary person that I'm lucky enough to spend my life with.

Contents

Nomenclature

- $P^1, ...P^{N_P}$ the entry or exit points of the TMA
- F^i the expected traffic flow on the entry/exit point P^i
- $\mathscr{R}^i$ an air route connecting P^i to the runway
- γ_h^i the projection of route $\mathscr{R}_i$ on the horizontal plan
- γ_v^i the vertical profile of route $\mathscr{R}_i$
- $\underline{z}^i(s)$, $\overline{z}^i(s)$ the minimal and maximal altitudes that an aircraft can attain at distance s from the starting point of route $\mathscr{R}_i$
- C the *center*: the starting point of the SIDs or ending point of the STARs for one graph structure
- $\mathscr{L}_1, \dots \mathscr{L}_{N_\mathscr{L}}$ the layers for a given graph structure
- N the number of vertices on one layer
- V the set of all vertices of a given graph
- $V_i = \{v_{i,j}, j \in \{1, \dots N\}\}$ the set of vertices on layer $\mathscr{L}_i$
- E the set of all edges of a given graph
- $e^i_{j,k}$ the edge that connects $v_{i,j}$ and $v_{i+1,k}$
- e a random edge, and $l(e)$ the length of e
- $\gamma_h[i]$ the edge $e^i_{j,k}$ belonging to the horizontal profile γ_h
- $\gamma_h[l_i, l_j]$ the portion of the horizontal profile that starts at layer $\mathscr{L}_i$ and ends at layer $\mathscr{L}_j$
- $\mathscr{L}_{ij} = \max\left\{l \in \{1, ..., N_\mathscr{L}\} \,\middle|\, \gamma_h^i[1,l] = \gamma_h^j[1,l]\right\}$ the merge layer between two routes i and j
- $l(\gamma_h) := \int_0^1 \|\gamma_h'(s)\|\, ds$ the total length of γ_h
- $d(t) = \int_0^t \|\gamma_h'(s)\|\, ds$ the *curvilinear abscissa* at $t \in [0,1]$
- $0 = \tau_1 < \tau_2 < ... < \tau_{N_\mathscr{L}} = 1$, such that $\gamma_h([\tau_m, \tau_n]) = \gamma_h[m,n]$

- $\widehat{e^{i-1}_{j,k} e^{i}_{k,l}}$ the angle formed by the edges $e^{i-1}_{j,k}$ and $e^{i}_{k,l}$. It can also be denoted $\widehat{v^{i-1}_{j} v^{i}_{k} v^{i+1}_{l}}$
- $\mathscr{O}$ the set of all obstacles
- $o = (B_o, l_o, u_o) \in \mathscr{O}$ an obstacle given by its base polygon B_o, lower and higher altitudes (resp. l_o and u_o)
- T the set of all cities
- $\tau = (B_\tau, \eta_\tau) \in T$ a city given by its location in the plane as a polygon B_τ, and the density function $\eta_\tau : B_\tau \to \mathbb{R}^+$ that gives the population density at a given point in the city
- $\alpha_{\min}$ and $\alpha_{\max}$ the minimum and maximum climb slopes
- d_h and d_v respectively the minimum horizontal and vertical distances to keep with an obstacle or another route
- d_{m} the minimum distance to keep between two merge points
- $\theta_{\min}$ the minimum angle between two routes at a merge point
- $\theta_{\max}$ the maximum authorized turn angle
- $n^{LF}_{\max}$ the maximum number of level flights
- $l^{LF}_{\min}$, $l^{LF}_{\max}$ the minimum and maximum length of a level flight

Glossary

- **ADS-B**: Automatic Dependent Surveillance - Broadcast
- **ANS**: Air Navigation Services
- **ATC**: Air Traffic Control
- **ASM**: AirSpace Management
- **ATM**: Air Traffic Management
- **AWY**: AirWaY
- **B&B**: Branch & Bound
- **CAT**: CATegory
- **CCO**: Continuous Climb Operations
- **CDG**: Charles-de-Gaulle airport
- **CDO**: Continuous Descent Operations
- **CFL**: Cleared Flight Level
- **CPDLC**: Controller-Pilot Data Link Communications
- **CTA**: ConTrolled Airspace
- **CTR**: Controlled Traffic Region
- **DME**: Distance Measuring Equipment
- **FAF**: Final Approach Fix
- **FAP**: Final Approach Point
- **FIR**: Flight Information Region
- **FMM**: Fast Marching Method
- **FMS**: Flight Management System
- **GA**: Genetic Algorithm
- **GNSS**: Global Navigation Satellite System

- **GPS**: Global Positioning System
- **IAF**: Initial Approach Fix
- **ICAO**: International Civil Aviation Organization
- **ILS**: Instrument Landing System
- **IMU**: Inertial Measurement Unit
- **INS**: Inertial Navigation System
- **IP**: Integer Programming
- **LOC**: LOCalizer
- **MAPt**: Missed Approach Point
- **MILP**: Mixed Integer Linear Programming
- **NDB**: Navigation DataBase **or** Non-Directional Beacon
- **PANS**: Procedures for Air Navigation Services
- **PBN**: Performance Based Navigation
- **PRM**: Probabilistic Roadmap Planner
- **RNAV**: aRea NAVigation
- **RNP**: Required Navigation Performance
- **RNP-AR**: RNP-Authorization Required
- **RRT**: Rapidly exploring Random Tree
- **RVR**: Runway Visual Range
- **RWY**: RunWaY
- **SA**: Simulated Annealing
- **SESAR**: Single European Sky ATM Research
- **SID**: Standard Instrument Departure
- **STAR**: Standard Terminal Arrival Route
- **TACAN**: TACtical Air Navigation system
- **TCA**: Terminal Control Area
- **TLS**: Transponder Landing System
- **TMA**: Terminal Maneuvering Area
- **TOC**: Top Of Climb

- **TOD**: Top Of Descent
- **TSP**: Traveling Salesman Problem
- **UAV**: Unmanned Air Vehicle
- **VOR**: VHF Omnidirectional Range
- **UIR**: Upper Information Region

Introduction

Air transportation is one of today's quickest ways to travel. The first commercial flight took place on January 1st, 1914 and paved the way for air travel as we know it today. The sector plays an important role in the global economy: in 2017, 4.1 billion passengers traveled by air on 45,091 routes globally, generating $2.7 trillion (3.6% of the world's global economic activity) and supporting over 65 million jobs worldwide [1]. The sector is growing every year, and forecasts see the number of flights in Europe increase by 53% between 2017 and 2040 [2]. Air transportation also plays a major role in the social landscape worldwide. As an example, in 2017, 57% of international tourists traveled by air [1].
The importance and size of global air traffic are expected to grow in the next 20 years according to EUROCONTROL's forecasts [3]. Over that period, traffic would increase by an average of 1.9% per year, and according to Boeing's forecasts (in [4]), the major growth should take place in China and regions like South and Southeast Asia as these countries develop quickly and more of their people begin to travel. Such an increase in demand is bound to bring new and complex challenges. Among them, the need to reduce carbon emissions, as there is more and more demand for greener transportation. In 2017, 859 million tonnes of CO2 were emitted by airlines, 2% of the global man-made emissions [1]. Companies and governments have already taken action on the matter, and CO2 emissions per passenger per kilometer have been halved since 1990 [1].
Another challenge to face is the already preoccupying congestion of major airports, due to a demand in air or ground movements higher than what the network is able to absorb. London - Heathrow is a good example, as it operates at 99% capacity, while other major European airports are around 65/70% [3]. As global traffic will very likely continue to grow, more and more airports will have to deal with congestion. According to EUROCONTROL's forecasts, 17 European countries could face a demand higher than their capacity in 2040, in the most likely scenario [3]. This increase in demand will lead to more delays, especially those related to Air Traffic Flow and Capacity Management (ATFCM), which could be multiplied by 5 in average over this period. In order to manage flights in the best possible way to reduce delays, several elements are necessary, like efficient infrastructures and sectoring of the airspace. In order to face these challenges, new systems are being developed, encompassing all aspects of air transportation. These systems are developed under the name SESAR (Single European Sky ATM Research) in Europe [5], and NextGen in the USA [6]

The airspace surrounding the airport is called Terminal Maneuvering Area (TMA). It is designed to handle the departing or arriving traffic and plays a critical role in ATFCM. Thus, increasing the capacity of the TMA is an essential step towards a sustainable growth of air traffic. In order to guide the aircraft from the airport to the en-route sector, or the other way around, the TMA contains pre-defined air routes: the procedures. They are respectively called Standard Instrument Departures (SID) and Standard Terminal Arrival Routes (STAR) and are created manually by experts as of today. This work requires to take many constraints into account, often making it impossible for humans to optimize any criterion in the design. For instance, creating short routes could help reduce the fuel consumption and CO2 emissions, making the topic a matter of interest, both economically and environmentally.

In this work, the design of SIDs and STARs is addressed in an automatic and optimal way in regard of the number of movements that can be undertaken in a given period of time. Several constraints are taken into account, such as ground obstacles, route separation, controllers' workload, military zones or cities. It features an adaptation of the Bellman-Ford algorithm to the problem, used in combination with the Simulated Annealing metaheuristic. The method presented in this document allows to take a certain number of operational constraints into account while providing an operationally efficient way to manage the merging points between the routes, while keeping a relatively low computation time in regard of the context.

This thesis is divided into five chapters. In chapter 1, the context of the study is given. We introduce the structure of the airspace and the tools used to achieve a safe and efficient air traffic management; the procedures are also introduced. In chapter 2, we present a review of the literature, on general methods for paths and trajectory optimization, but also on more specific methods used in the context of air transportation. We put an emphasis on a specific work that serves as a basis of comparison later in the thesis. Chapter 3 presents the way that was chosen to model the problem as an optimization problem. We give the way in which we model the search space as well as the constraints and the objective function. In chapter 4, the resolution approach that we developed is introduced in details. We explain the way in which the constraints and objective were taken into account, as well as some algorithmic procedures and the limitations associated to them. Finally, in chapter 5, the results obtained with our method are presented. Four instances were chosen to conduct the tests, three of them taken from the literature. A comparison with these results allow to estimate the achievements and margin for progress of our work. The analysis of the results of the tests allow to have a better understanding of the underlying mechanisms of our method.

Chapter 1

Problem context: the framework of procedure design

In this chapter we define and explain the current state of Air Traffic Management (ATM): the stages that allow for an efficient management of the flights in due time, the various steps of a typical flight, and the specificities of each step, as well as the regulatory framework in air traffic. We also introduce some of the tools used in air navigation and the ongoing changes made to these equipments. Finally, we specifically present the object of this thesis, the procedures: what they and their purpose are, and the critical aspects of their design.

1.1 The current and future state of air traffic management

ATM is a large field that encompasses all systems (humans and equipment) that allow aircraft to take off from an airport, travel to their destination and land safely. According to the Commission of the European Communities [7], it covers three main services:

- *Air Traffic Control (ATC)*: its purpose is to ensure sufficient separation between aircraft in the air and on the ground, and between aircraft and ground obstacles so as to avoid collisions. This has to be done in a way that keeps an orderly flow of air traffic.
- *Air Traffic Flow Management (ATFM)*: its purpose is to regulate the flow of aircraft to avoid congestion in busy sectors. It aims at matching in the best possible way the supply to the demand in terms of airspace capacity by anticipating the needs in controllers.
- *AirSpace Management (ASM)*: its purpose is to define and allocate the airspace sectors as efficiently as possible between the different actors (civil and military).

These operations take place at different moments in time. For instance, the design of the airspace sectors has to be done before the scheduled flights

take place. To distinguish between these temporal milestones, three levels of planning are in use in the aviation sector:

- The *strategic level* encompasses all operations that must be done as early as up to one year before the actual flights, and until a few weeks before them. Such operations can be to build an estimate of the demand, congestion or other indicators, or to design the nominal routes that aircraft will follow on a daily basis.

- The *pre-tactical level* encompasses all operations that take place between a few weeks and one day before the actual flights. In this phase, more precise data is collected, such as actual demand in air traffic or weather, in order to adjust the work previously done at the strategic level.

- The *tactical level* encompasses all operations that take place the day of the flights: traffic control, possible re-routing, departure time slots allocations and so on.

In order to manage the traffic, especially at the tactical level, various equipments and systems are used and regularly improved, such as:

- The *Flight Management system (FMS)*. Embedded in (almost) all aircraft, its purpose is to allow the on-board staff to pilot efficiently. It gathers all the relevant information for the flight, such as the flight plan, the trajectory to follow and a way to transfer it to an automatic pilot, a *Navigation DataBase (NDB)*, which usually contains:

  - the airways;
  - the waypoints, which will be detailed later in this document;
  - the airports and runways;
  - the departure and arrival routes (SIDs and STARs), which will be explained later in this document.

- The *surveillance system*, which gathers all the technologies used to locate aircraft and monitor traffic. As of today, this is done with the use of different kinds of radars. The primary radar only detects the presence of objects in its surrounding airspace, and works on its own. The secondary radar establishes a connection between the ground and the aircraft, and allows the transfer of more information depending on the used *mode*. The mode of the radar determines the nature of the exchanged information. The mode A transmits the aircraft's identifier. The mode C, as the mode A, transmits the aircraft's identifier, but also its altitude. Finally, the mode S allows to transfer any kind of information, such as the aircraft's identifier, altitude, heading, speed etc. The information is then displayed to the controllers.

- The *communication system* between pilots and controllers. Currently almost exclusively done by voice over radio, it could be improved by exchanging automatic messages with the ground. This solution, called

Controller-pilot data link communications (CPDLC), is already in use in several control centers.

These systems and technologies are bound to be improved, since they are insufficient to face the upcoming increase in traffic demand. Therefore, new projects are emerging, such as the NextGen project in the USA or SESAR (Single European Sky ATM Research) in Europe. These projects aim at improving the capabilities of ATM. For instance, instead of using radars to locate and acquire data about flights, the information could be transmitted directly by the aircraft with the ADS-B (Automatic Dependent Surveillance - Broadcast) technology, which uses a GPS positioning and is therefore more accurate and reliable than the technology currently in use. Significant improvements are also made in the field of air navigation with the introduction of Performance Based Navigation (PBN), which will be detailed later.

1.2 Airspaces and flight structure

In order to distribute the traffic in the most efficient way, the airspace has to be segregated into different *sectors*. In this section, we are going to review the different types of sectors and their properties, as well as the standard progress of a flight.

1.2.1 The different airspaces and their purpose

In order to achieve efficient aircraft directing and separation, the airspace is generally divided into different layers:

- The ground layer at an airport is a layer on its own. It is not usually referred to as an airspace, but it still has dedicated controllers on a large majority of airports. Its purpose is to allow controllers to handle traffic that go back and forth the gates and runways in a safe and efficient way.

- The *control zone*, or *Controlled Traffic Region* (CTR) is a controlled portion of airspace that extends from the ground to a specified height. It is designed to protect the traffic that take off and land onto the airport.

- The *control area* (CTA) is a volume of controlled airspace that exists near an airport, with specified nonzero lower and upper limits. Usually, a CTA is located directly above one or several CTR (when, for example, several airports are close to one another), but exceptions exist. It is designed to provide a controlled portion of airspace that makes a link between the upper limit of a CTR and the airways.

- The *Terminal Maneuvering Area* (TMA), or *Terminal Control Area* (TCA) in the USA and Canada is a term designating a particular type

of CTA, which surrounds large airports (it often covers several airports close to each other). Usually, this part of airspace is divided into several cylindrical layers of increasing radii, forming an "upside down wedding cake" above the CTRs (see fig 1.1). This type of airspace is designed to handle heavy traffic between the upper limit of the CTRs and the airways.

- The *airways*, or *air routes*, are defined corridors that connect two geographical points, identified either by satellite coordinates or by navaids (physical devices on the ground that aircraft can detect and fly to). They have a specified altitude and width and allow aircraft to transit.

- The *Flight Information Region* (FIR) is a portion of airspace in which a flight information service and an alerting service are provided. It is the largest regular division of airspace, and can cover an entire country. Sometimes, a FIR is divided into two portions. In these cases, the upper portion is called *Upper Information Region* (UIR).

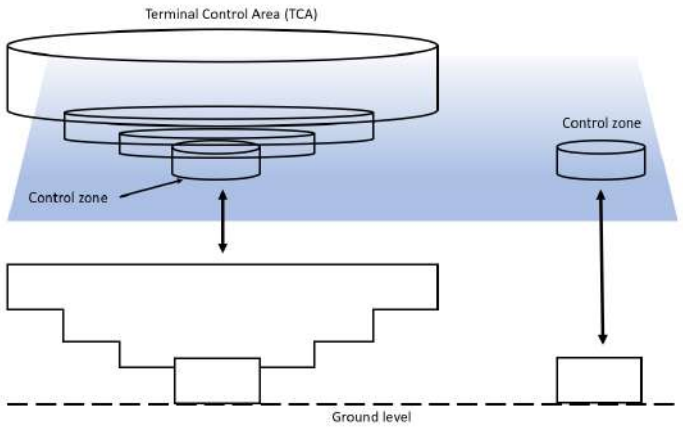


Figure 1.1 – The simplified representation of a TMA.

These regions of airspace are often divided into categories, identified with letters, according to the requirements an aircraft and its crew must meet to be authorized to enter them. Some portions of airspace are not controlled at all. Finally, some portions of airspace can be temporarily or permanently restricted. These are regions such as:

- *Restricted areas*, in which military activities can be conducted (missiles, artillery firing...). These areas may be flown across when they are inactive, if a clearance has been issued by the controllers. They are identified on the charts by the letter R.

- *Warning areas*, that are roughly the same as restricted areas. They exist in the USA and extend to 3NM outwards the coasts. They are identified by the letter W.

- *Prohibited areas*, in which all flights are prohibited as a matter of national security. They are often located above major cities or sensitive ground areas. They are identified with the letter P.

- *National security areas*, which are located above areas where security and safety of ground facilities are necessary. It is required to voluntarily avoid these areas. They are not identified with a specific letter.

- *Temporarily restricted areas*, which may be used to clear the airspace to allow important or urgent traffic, such as rescue aircraft, to transit quickly.

The sectoring of the airspace is necessary to divide the workload between controllers and thus achieve a safe and efficient transit of all aircraft.

1.2.2 The different steps of a flight

A typical flight is divided into three main steps: the climb, the cruise and the descent. When a problem occurs in the landing phase, a fourth step is added: the missed approach. Here, we explain in details how these steps are performed and when. Figure 1.2 provides an illustration of the different steps of a flight.

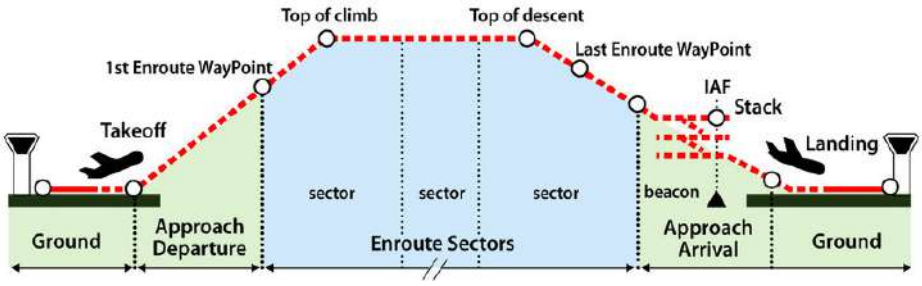


Figure 1.2 – The different steps of a flight (taken from [8]).

1.2.2.1 The climb

This phase is the first to take place. In order to take off, a strict procedure must be observed:

- The *push back* phase: when the plane is finished boarding, the pilot asks the ground controller for a clearance to be pushed back from its

departure gate. A departure time slot is issued for this aircraft, and the pilot completes the checklist before asking a clearance for taxi.

- The *taxi* phase: it is one of the most complex phases to handle for a controller on large airports. It consists in "driving" the plane from a starting point to an ending point on the ground, by using the *taxiways*. In major airports, it is one of the main causes of congestion, since many aircraft need to transit, mostly between gates and runways, and the taxiways network is not always able to absorb all the demand. Some research is conducted to improve the path finding and reduce congestion on taxiways ([9] [10]).

- The *take-off* phase: once an aircraft has taxied to its assigned *holding point*, which is located next to the runway (note that a holding point can also be encountered at each road crossing) (see fig. 1.3), the pilot can ask for a take-off clearance. When it is granted, the pilot is authorized to go onto the runway and take off. During this phase, several moments are distinguished (see fig. 1.4).

Figure 1.3 – Two planes waiting at a holding point.

  - The plane accelerates until it reaches a speed V_1. Past this point, it is no longer safe to abort the take-off.
  - The plane keeps accelerating until it reaches a speed V_R, the *rotation speed*, at which the nose is lifted from the ground.
  - The plane reaches the speed V_{LOF}, the lift-off speed, at which it no longer touches the ground.

– The plane reaches the speed V_2, which is the minimum speed at which the plane can safely climb with one engine inoperative.

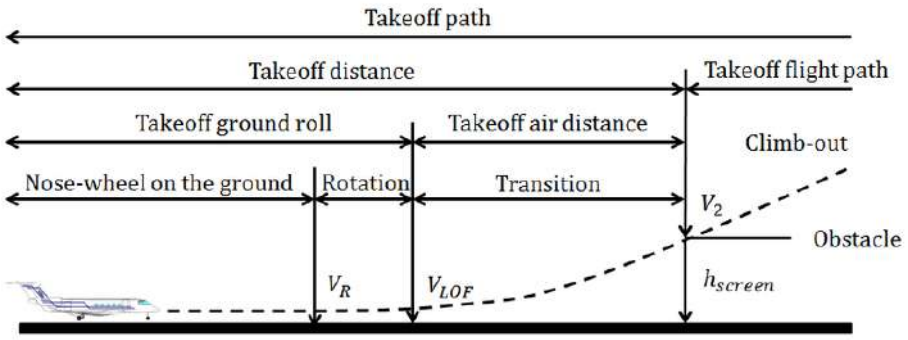


Figure 1.4 – The phases of take-off (taken from [11]).

- The *climb* phase: once the aircraft has taken off, it has to reach the en-route sector. It is almost always done by following pre-designed air routes called Standard Instrument Departures (SID) through the TMA. The structure and properties of a SID will be detailed in the next section.

1.2.2.2 The cruise

In this phase, the aircraft has passed its *Top Of Climb* (TOC) point, at which it stabilized its altitude, and travels towards its destination. The routes are straighter in average than in the previous phase, and the airspace less congested. However, with the increasing growth of air traffic, this situation is bound to change. The main difficulty for controllers in this phase is to anticipate and resolve conflicts between crossing aircraft. The subject is widely researched, and many works have already been published on the matter, with different ways to avoid conflicts: change the heading of the aircraft [12] [13], its altitude [14] [15], its speed [16] or holding it on the ground even before it takes off [12] [17].

1.2.2.3 The landing

In this phase, the aircraft is close enough to its destination to initiate a descent. After obtaining a clearance from the controllers, the pilot leaves the airway and enters the TMA of the destination airport. As of today, similarly to the climb phase, the descent is carried out in several phases. Usually, there are three of them. The first phase is the *descent*, in which the aircraft starts to lose altitude (this point is called the *Top Of Descent* (TOD) and reaches a first control point called the *Initial Approach Fix* (IAF). This phase may include holding paths when the airspace is congested. The second phase is the *approach*. Here, the aircraft is ready to land and travels from the IAF to the last control point before the runway, denominated *Final Approach Fix* (FAF) or *Final Approach Point* (FAP). In this phase, a plane will have its flaps out and will deploy the landing gear. The final step is the *landing*

on the runway and taxi to the arrival gate. Current works aim at reducing or even suppressing the need for holdings, level flights, or radar vectoring (i.e. when a controller temporarily deviates an aircraft from its scheduled route in order to avoid collisions). When arriving at large airports, aircraft usually follow pre-designed air routes through the TMA. Such routes are called *Standard Terminal Arrival Routes* (STAR). Note that, depending on certain parameters (such as the weather and the aircraft capabilities), the TOD can be located well after the beginning of the STAR, which means that an aircraft may engage in a STAR while still being in its cruise phase. The characteristics and purpose of a STAR will be detailed in the next section. Once it has reached the end of the STAR and has been cleared to land, the aircraft may finish its descent onto the runway and taxi to its assigned gate. In order to land safely, the pilot must ensure that all conditions are met. When it is not the case, they must abort the landing. This is called a *missed approach*, that will be detailed in the next paragraph. The last possible moment for a pilot to decide whether to land or not is directly related to the *decision height* and the *Runway Visual Range* (RVR), which is the distance over which a pilot is able to identify the center line of the runway. These values depend on three parameters:

- The equipment available at the airport;
- The equipment of the aircraft;
- The training of the pilots.

Depending on them, the options for precision landing are classified into *categories*:

- Category I: The decision height is 200ft (60m), with an RVR of 550m (1800ft);
- Category II: The decision heigth is 100ft (30m), with an RVR of 350m (1200ft);
- Category III A: The decision heigth is 100ft (30m), with an RVR of 200m (700ft);
- Category III B: The decision heigth is 50ft (15m), with an RVR of 50m (150ft);
- Category III C: There are no limits, the pilot can land without any visibility.

1.2.2.4 The missed approach

Under certain circumstances, an aircraft that was cleared to land may have to abort the landing and re-take off. This situation can be caused by various factors: debris or another aircraft on the runway, insufficient visibility at the decision height (which is the minimum height at which a pilot can decide whether to land or not, and depends on the category of landing) or too much

wind too close to the ground... Depending on the cause, the aircraft either attempts to land another time, or is directed towards another airport (in the case of persisting bad weather, for instance).

1.3 The purpose and design of SIDs and STARs

To be able to handle safely heavy traffic departing from and arriving to large airports, a specific type of air routes has been created: the *procedures*. They indicate the path that an aircraft should take in order to proceed from the runway to the en-route airspace (or the other way around). The general regulatory aspects for air traffic and management are the topic of one of the International Civil Aviation Organization (ICAO)'s Annexes. The procedures to follow for a given topic in aviation are gathered in the Procedures for Air Navigation Services (PANS) documents, such as [18]. In this section, we are going to describe how the procedures are made, and for which types of aircraft.

1.3.1 The navigational aids and instruments

In order to design a procedure, one must know beforehand the type of aircraft that will use it, as well as the capabilities of the airport under consideration with regard to its take-off and landing equipment. Here, a non-exhaustive list of equipments is presented, along with their purpose and performances.

- The *Distance Measuring Equipment (DME)* is an equipment that measures the distance between an aircraft and a ground station by using radio navigation technology. It works by timing propagation delays of radio signals. In order to work, there must not be any obstacle on the line of visibility from the station to the aircraft (so it cannot detect, for instance, aircraft below the horizon, or behind a mountain). This equipment is often used with an azimuth guidance system, like a VOR or a TACAN.

- The *VHF Omnidirectional Range (VOR)* is a short-range radio navigation system that allows aircraft with an adequate receptor to know their bearing (angle) relatively to the station. It works by emitting two signals: a "master" signal as reference, and another signal whose phase differs from the master's by the value of the angle.

- The *Tactical Air Navigation System (TACAN)* is equivalent to the combination of a VOR and a DME, with more accuracy. It is mostly used by the military, but can also be used by civil aircraft.

- The *Non-Directional Beacon (NDB)* is a radio transmitter that has the same purpose than a VOR, which is to provide an aircraft's bearing. It works by emitting a signal that is received by the aircraft, that in turn analyzes it to determine the direction in which it is the strongest.

This direction is that of the NDB. In opposition to the VOR, it follows the curvature of the earth, so it can be perceived from a longer range at a lower altitude. It is however sensitive to atmospheric conditions, mountains or coast reflection.

- The *Localizer (LOC, or LLZ)* is a part of the Instrument Landing System (ILS) equipment of an airport. Located beyond the runway, it provides horizontal guidance to the aircraft and allows them to stay in its axis by giving the horizontal deviation from it.

- The *Glide* is also part of an airport's ILS. It is the vertical pendant of the Localizer, as it provides the optimal slope of descent to the runway to the aircraft.

- The *Transponder Landing System (TLS)* is a precision landing system that can be implanted where a LOC/Glide equipment would be ineffective, for example in uneven terrains (mountains) or terrains with large obstacles nearby (buildings, for example). It works by using three antennas that interrogate the aircraft's transponder and uses the information to perform a trilateration.

- The *Inertial Navigation System (INS)*, sometimes referred to as Inertial Measurement Unit (IMU), is an embedded system that allows an aircraft to compute its position, speed and orientation without any exterior equipment, by referring to the aircraft's last known parameters and by using a computer, accelerometers and gyroscopes.

- A *Global Navigation Satellite System (GNSS)*, such as the GPS (from the USA), GLONASS (from Russia), Galileo (from Europe) or Beidou-2 (from China), is a satellite-based radio navigation system that uses trilateration to provide its position, speed and time to a receiver with a great accuracy. For the system to work, the receiver must be "visible" from at least four satellites.

As we established, these equipments have varying accuracies, and thus cannot be all used in the same way.

1.3.2 The different types of procedures

In the air transportation domain, the procedure design is a highly regulated topic, that is subject to many changes and improvements. Due to the complexity of the task, designing the procedures is a task entirely handled by human experts by hand. In the following paragraphs, we will go over the precise definition of SIDs and STARs, and the different possibilities that exist to design them.
A **Standard Instrument Departure** (SID) is a procedure, a plan of operations that an aircraft equipped with IFR (Instrument Flight Rules, which means that the aircraft can be piloted by relying on the instruments) has to follow in order to depart from an airport. It begins right after the take-off and leads to the en-route sector. The pilot must obtain a

clearance from the controllers to fly it. A **Standard Terminal Arrival Route** (STAR) is the landing pendant of a SID. It usually connects the end of the en-route sector to the IAF, which marks the transition to the approach control area, or sometimes all the way to the FAF, the last point before the runway's threshold. A procedure usually contains information and requirements about altitude at certain points. They usually include one or several level flights. However, a possible way to fly the procedures is to perform **Continuous Climb Operations** (CCO) and **Continuous Descent Operations** (CDO), which allow to climb and descend without the need for an intermediary flight at a constant altitude (see fig. 1.5). This improvement allows to save fuel as well as to improve the noise abatement in the vicinity of airports. As these operations require additional margins, they tend to not be used in heavy traffic conditions. It can be seen that the

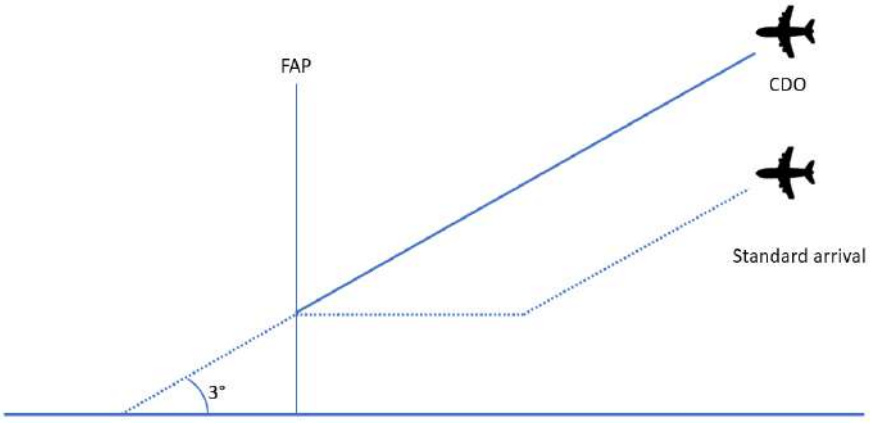


Figure 1.5 – The illustration of Continuous Descent Operations.

SID and STAR design falls into the field of path planning, which is not to be mistaken with trajectory planning. This difference will be explained in the next chapter.

1.3.2.1 The conventional procedures

The conventional procedures rely solely on the ground-based equipment. They are often denominated by the equipment they use. Therefore, the type of a STAR could be VOR/DME, or DME/DME for instance. These procedures are constructed by using segments between the successive devices on the ground. Their precision is relatively low compared to more recent technologies. Typically, a DME has a minimum error range of $\pm$ 400m, and increases with the distance to it. As a result, wide additional margins must be taken when designing a conventional procedure in order to ensure the safety of the aircraft. The use of the airspace is then greatly limited. Another consequence is that the aircraft flying conventional procedures are very likely to be scattered around the nominal path, since their equipment lacks accuracy. This often leads the controllers to give heading directions to manage the traffic, and creates congestion.

1.3.2.2 The RNAV procedures

The *Area Navigation* (RNAV) belongs to the wider category of the *Performance Based Navigation* (PBN). As opposed to conventional procedures, which are defined by the equipment used, PBN is defined based on its operational requirements. Therefore, as long as the required performances are met, an aircraft flying with PBN can rely on existing equipment, such as VOR or DME. The RNAV subcategory of PBN is characterized by the ability to fly any path, defined by *waypoints*, which are geographic fixes such as ground navaids but also points given in lat/long coordinates, for instance (see fig 1.6). Various denominations are in use

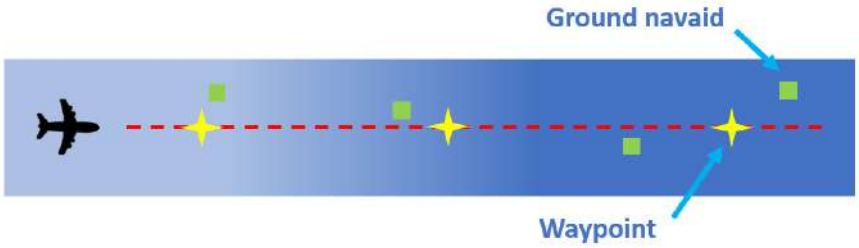


Figure 1.6 – The illustration of waypoints.

regarding RNAV, each corresponding to the degree of accuracy available. More specifically, it gives information about the lateral accuracy that the system is expected to achieve during at least 95% of the flight. Thus, a system qualified for "RNAV 1", for instance, will be able to remain within a 2NM-wide path centered on the desired track (1NM on each side) for at least 95% of the flight. The usual systems are RNAV 10 for the en-route part of the flight, and RNAV 5, 2 or 1 for the terminal area. The RNAV-qualified aircraft must also comply with requirements of continuity (the service must not be interrupted during the flight).

1.3.2.3 The RNP procedures

The *Required Navigation Performance* (RNP), like the RNAV, belongs to the PBN category. It was first introduced by the ICAO in 1998 in the reference document 9613 [19]. As RNAV, it is referred to as "RNP X" where X is the acceptable lateral error. The major difference with RNAV is that RNP is required to provide integrity (the information provided by the equipment must be reliable) by on-board performance monitoring and alerting. This component must be able to detect a failure that prevents the aircraft to remain within two times the RNP value, or if the probability of this happening exceeds 10^{-5}.
A specific type of RNP is defined, the RNP AR, for Authorization Required [20]. This type of navigation is even more precise than standard RNP, with RNP values going from 0.3 to 0.1. They are used only for the approach part of a flight, and require a dedicated authorization given by the corresponding safety authority. These enhanced capabilities allow a wider range of maneuvers in the vicinity of the runway when landing. For instance,

with RNP AR, it is possible to go from the FAP to the decision altitude with a curve instead of a segment in the axis of the runway. Figure 1.7 provides an illustration of the comparison between conventional, RNAV and RNP capabilities.

In all cases, procedures must include sufficient margins on each side

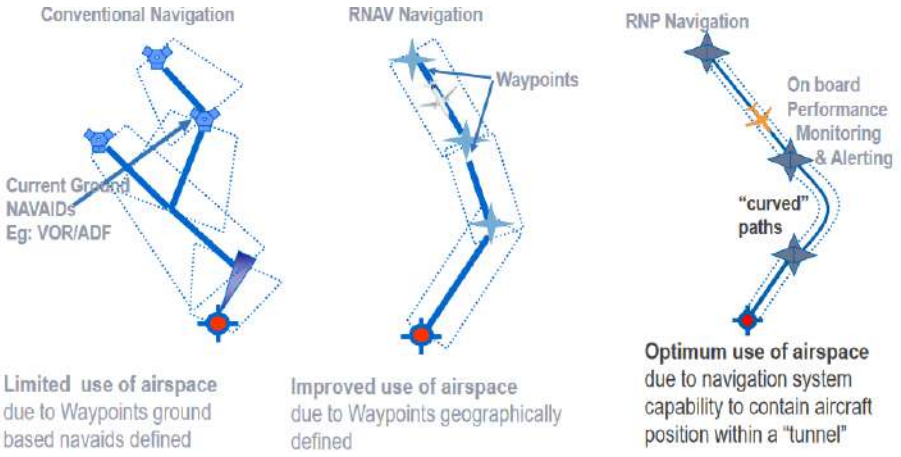


Figure 1.7 – The comparison between conventional, RNAV and RNP procedures (taken from [21]).

of the desired path, and between the path and the terrain or buildings in altitude. Each type of procedure has its specificities, and each one of them has been covered in a dedicated document from the ICAO. Since conventional procedures are qualified by the equipment they use (and not their performances), a manual is issued for each combination of equipments: VOR/DME, DME/DME... (example in [22]). The way to build these protection areas will not be addressed in this thesis due to the complexity and specificity of the topic. The reader can refer to the official documentation (Annexes and PANS documents) for further reading. An example of published procedure for Paris Charles-de-Gaulle airport is provided in fig. 1.8.

1.3.2.4 A particular structure: the Point Merge

In order to decrease congestion, a new concept has emerged, first introduced by EUROCONTROL in 2006 [23]: the *point merge*. In the vicinity of large airports, there are multiple STARs to merge together and controllers must often temporarily deviate the aircraft from their route to manage their sequencing (this is called *vectoring*). They can also make them gain or reduce speed, although vectoring is preferred. To alleviate the workload that this situation generates for the controllers, the point merge works as follows (see fig 1.9):

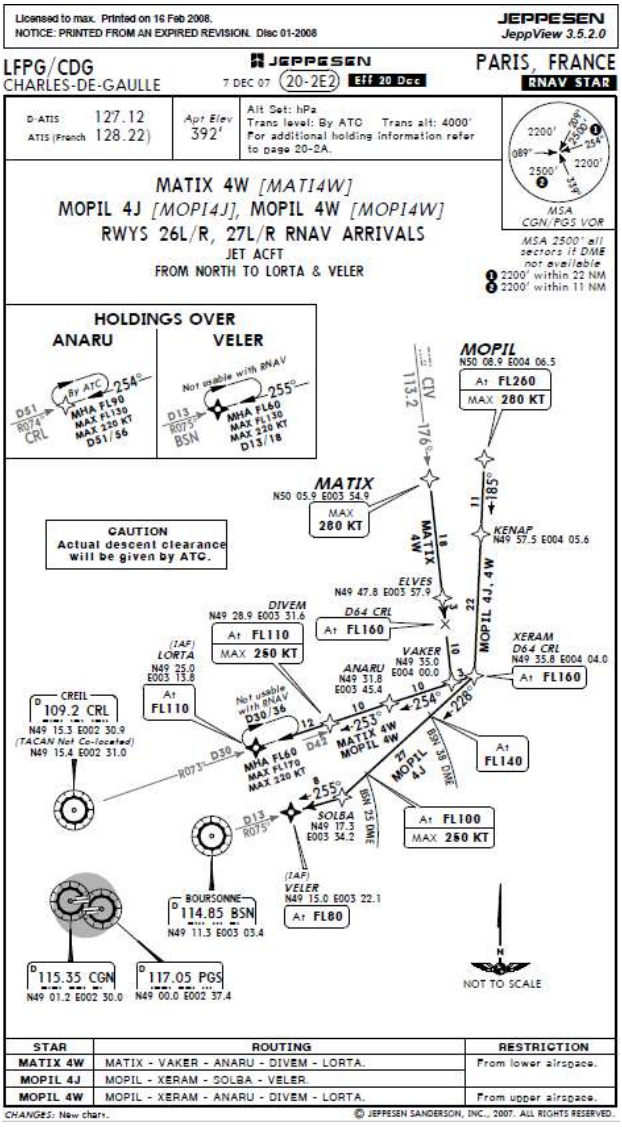


Figure 1.8 – An example of published procedure at CDG airport.

- A waypoint is set as the point merge. It is the point where the two STARs are supposed to merge initially.
- A cone is created, starting from the point merge and expanding towards the beginning of the STARs.
- Layers are created inside the cone. They are portions of circles centered on the point merge, set at different distances from it. Each distance corresponds to a level of congestion: the higher the congestion, the greater the distance.
- When arriving to the specified layer while flying the STAR, the aircraft automatically set their route so as to remain on the layer until instructed otherwise.
- When the controller instructs it, the aircraft may reset its route to fly directly to the point merge.

This method allows for a naturally ordered sequencing, way easier to manage than a vectoring situation, and has already been implemented in 25 locations around the world. The design of departure and arrival procedures is a

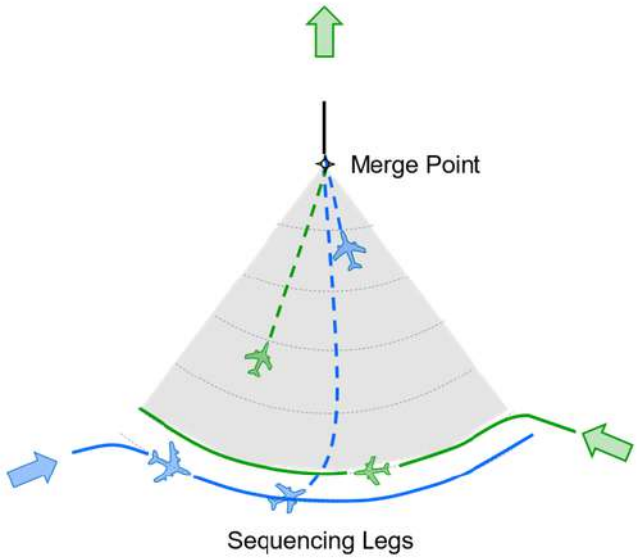


Figure 1.9 – The illustration of the Point Merge concept [24].

very complex topic, due to the number and specificity of the constraints, especially the design of protection areas around the paths. This specificity causes each case to be different and generic approaches to be very likely to fail. The method presented in this work allows for exploring a large number of possible designs. By doing so, it provides the possibility to introduce optimization criteria, which is not easy when the design is done by hand. Our method is designed to remain as close as possible to the way in which the design is handled as of today.

1.4 Operational context and objective of this thesis

This thesis is sponsored by CGX Aero (https://cgx-group.com/fr/index.htm) as one of their research projects. CGX is a company providing solutions for airport and procedure design as part of their activities. In this context, the aim of our work is to provide a tool that is capable of designing SIDs and STARs in an automatic way, including in complex situations (for instance: with many routes to design, or in a complex terrain). The result of our solution should be as close as possible to the design that an expert could make. This would allow the expert procedure designers to be given a first solution to work on and improve if needed in a very short time.
At CGX, a tool for checking the compliance of a given procedure with the separation and route protection rules is available: GéoTITAN®. This software allows to draw a procedure in a given terrain and test its compliance with the constraints. An example is provided in fig. 1.10. As our

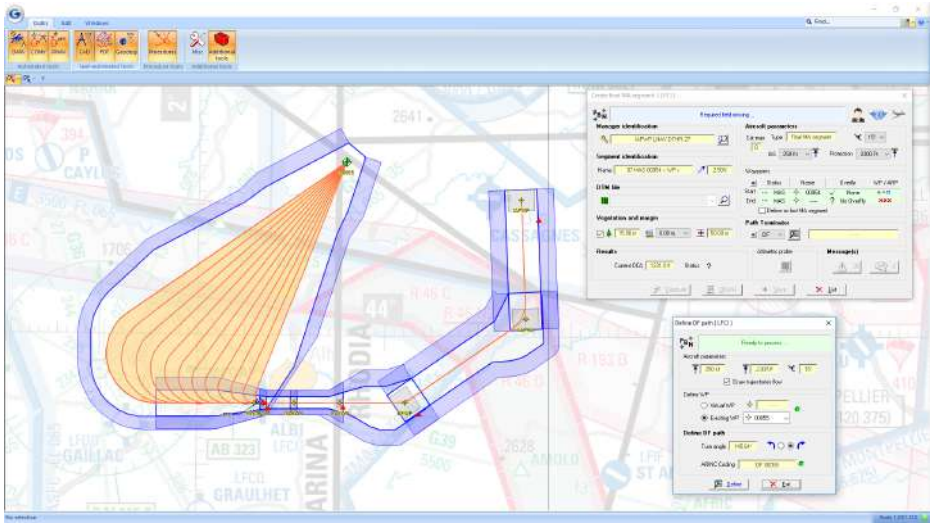


Figure 1.10 – The GéoTITAN® software.

work aims at designing procedures, its output could be interfaced with the input of GéoTITAN® to proceed with a thorough evaluation of the solution.

The objective of this thesis is to provide a method to automatically design SIDs and STARs by optimizing various criteria while taking into account the operational constraints, with the aim of improving on the drawbacks of existing works. Indeed, although the topic of path and trajectory design has already been widely researched, few works in comparison were dedicated to the specific topic of SID/STAR design. In the next chapter, we present some methods and algorithms that can be applied to solve the general problem of path and trajectory finding, but also some works that tackle the problem of automatic design of SIDs and STARs. Their specificities, advantages and limitations are emphasized.

Chapter 2

Literature review

In this chapter we present the work that has already been done on the topic of automatic procedure design, and more generally the works on automatic path or trajectory design. We define *paths* and *trajectories* as follows:

- A *path* is a curve that connects a starting and an ending point while avoiding static obstacles. The notions of time or speed are not taken into account, as only static elements are taken under consideration. As an example, a path can be assimilated to a railway.
- A *trajectory* is the association of a path and the way to follow it. In a trajectory finding problem, dynamic obstacles and speed must be taken into account. For instance, flying a plane in the presence of hazardous weather falls into the trajectory finding category.

In this thesis, the objective is to design paths, and although the specific topic of automatic procedure design has not yet been extensively explored, the more general problem of path and trajectory finding has been a subject to interest for several decades, particularly in the domain of robotics. In most of the related works, the search space is taken to be of dimension two. However, a certain number of works take the third dimension and the increase in difficulty in processing into account. This chapter is divided into three parts. In the first part, we present the methods relative to the representation of the search space and of the routes or trajectories. In the second part, a review of methods for solving a path or trajectory finding problem is given. In the last part, we focus on two specific SID and STAR optimization works that were taken as references for the tests in this thesis.

2.1 Search space and route representation

The first step in a path or trajectory planning problem is to identify the form to be given to a route, and therefore the way in which the search space is to be constructed. This section focuses on the methods used to represent these elements.

2.1.1 Triangulations

A standard way to discretize the space is to perform triangulation. It consists in partitioning the search space with triangles. In this section, we consider that the search space is given as the interior of a polygon, in 2D or 3D depending on the discussed case. Triangulation creates "tiles", thus making the path finding problem discrete.
The *Delaunay triangulation* [25] is probably the most frequently used method for triangulation, since it verifies in 2D a certain number of mathematical properties [26]. Among them, the fact that the circumscribed circle of each triangle contains only the points of this triangle. This requirement allows to maximize the minimum angle in each triangle, which yields a result without any "degenerate" cell (for example flattened cells). This type of discretization is achievable in a very reasonable time: $\mathcal{O}(n \log n)$, where n is the number of points used to describe the search space [27]. Another advantage of this method is that it is easily translatable to 3D [26]. It is then referred to as *tetrahedralization*.
The main drawback of this method is that in certain cases, the result is still not satisfactory: the shape of the search space makes it impossible to obtain a decent triangulation, as illustrated by fig. 2.1 where some triangles are too small, too thin and generally speaking very heterogeneous. This causes the need for another category of triangulation: the *Steiner triangulations*. They consist in adding new points on the border of the search space in order to obtain a more homogeneous triangulation (see fig. 2.1). The goal is then to choose the additional points, called *Steiner points* (as explained in [28]). The way to choose the points varies with the nature of the problem under consideration: the Steiner points will not be in the same number and place whether the aim is to obtain the most regular triangles, or to achieve a great precision in the discretization of the research space, for instance.
The notion of triangulation is closely related to the structures of graphs

(a) The Delaunay triangulation. (b) A Steiner triangulation.

Figure 2.1 – Examples of the Delaunay triangulation and a Steiner triangulation in [28].

or trees: once the space is divided into cells, each one of these cells can be assimilated to a node. It is then possible to apply path finding algorithms on this structure. The problem consisting in finding the minimal Steiner tree (i.e. a tree of minimal total length) is a NP-complete problem and is still actively researched as of today [29] [30].

2.1.2 Natural graph structures

Apart from triangulation, there are other methods that allow to discretize a search space. The most intuitive approach is probably the creation of a *visibility graph*. This method can be used when the goal is to compute one or several shortest paths between points in a space where there are polygonal obstacles [31]. Intuitively enough, the shortest path between two points is a straight line when there are no obstacle, or a set of segments that connect the starting and ending points by passing by the vertices of the obstacles (see fig. 2.2). Several works propose methods for finding the shortest path in a visibility graph, such as [31] [32] [33]. Note that this approach for a shortest path between two points is no longer acceptable in 3D, since the shortest path does not necessarily pass by the vertices of the obstacles. As an example, we can consider the case in which the starting and ending points are separated by a rectangular parallelepiped (see fig. 2.3). More on the subject can be found in [34]. The shortest path is likely to pass by one of the edges. Other works contribute to the extension of the visibility graph in 3D, such as [35] [36]. In some cases, the obstacles can be both polygons

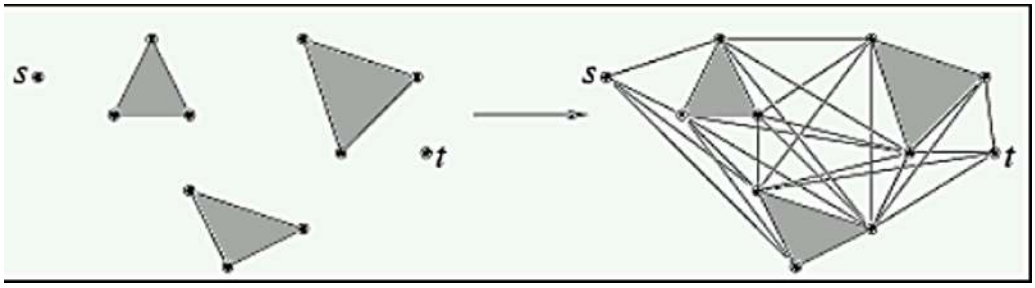


Figure 2.2 – An example of visibility graph.

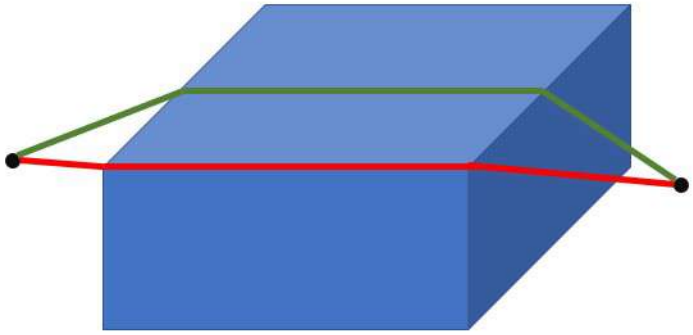


Figure 2.3 – The shortest path between the two points (in green) does not pass through a vertex of the parallelepiped. Therefore, connecting only the vertices is no longer sufficient to find the shortest path in 3D with a visibility graph, since the output would be the path in red.

and closed curves. In these cases, the equivalent to the visibility graph is

called the *tangent graph*, introduced by Liu and Arimoto in 1992 [37]. In this representation, a vertex of the graph is a point lying on the boundary of an obstacle. An edge in this graph is then either a segment joining points belonging to two different obstacles or a curve between two nodes of the same obstacle. This method can be used in cases where the obstacles to be avoided are circular, as in [38]. By using this method, the mobile passes as close as possible to the obstacles. In some cases, it is instead required to pass as far as possible from any obstacle. In order to achieve this goal, an intuitive behavior would be to follow paths that are at an equal distance from all obstacles, so as to remain "in the middle" of the unobstructed area. This notion of equal distance to objects is the principle of *Voronoi diagrams*. This technique relies on the concept of *influence area* of a particle, which allows to determine all paths that are as far as possible from all obstacles (see fig 2.4). The specificity of this diagram is that it is in fact the dual of the Delaunay triangulation. Thus, there exists a "natural" transition from one to the other: every particle in the Voronoi diagram is a vertex in the corresponding Delaunay triangulation. A bibliography on the subject can be found in [39].

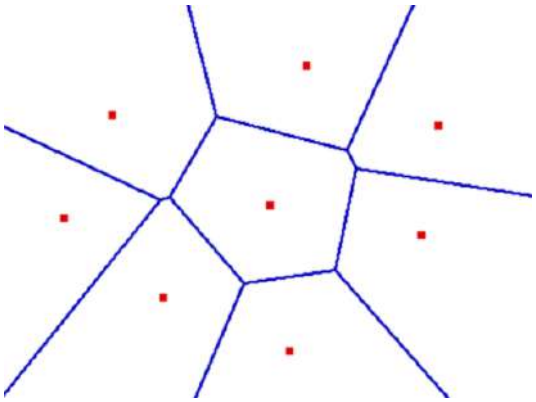


Figure 2.4 – An example of Voronoï diagram.

2.1.3 Cell decomposition

Another way to discretize space is to sample it by the means of a grid. There are several types of such structures. The *constant step grids* are probably the simplest to implement: the only thing to do is to choose the length of the side of the cells, which will remain fixed and is the same for all grid cells. The drawback of this method is that it is very approximate in the cases where obstacle detection is an issue, since the resolution is decided beforehand. Conversely, if a "good" resolution is chosen in order to accurately detect obstacles, this method is very likely to be too expensive in terms of data storage, since it will also yield a good resolution on empty areas. On the other hand, transposing this method to 3D is immediate. In order to solve the problem of constant resolution in constant step grids, one can use similar data structures, in a dynamical way: the *quadtrees* (see fig. 2.5). The idea

is to store only useful information, which means to require precision only in the vicinity of obstacles or points of interest. This method allows to store information on various levels: each square is divided in four, and so on until the required precision is attained on the interesting parts. This method yields a "zoomable" structure, which is very useful in managing various levels of precision. The transition to 3D is once again quite easy: instead of squares, the manipulated structures are cubes, and they are divided into eight parts. They are then called *octrees*.

Another way to subdivide the search space into cells can be, for instance,

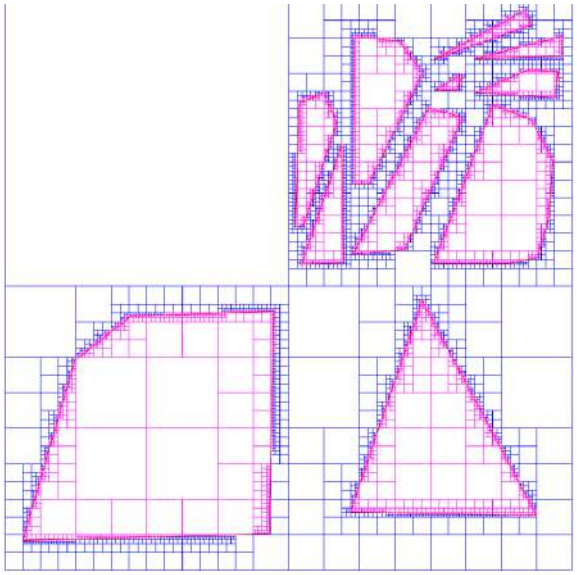


Figure 2.5 – An example of quadtree ([40]).

with parallel segments (see fig. 2.6). In this approach, each segment either passes through or ends on a different vertex of polygonal obstacles. Other techniques of space subdivision can be found in [41] [42]. A connectivity graph can then be established, by symbolizing each cell by a vertex and by connecting two vertices whenever the corresponding cells are adjacent. A path in such a structure can then be transposed from the graph to the cells by taking each time the middle of the segment between two adjacent cells as a passage point, for instance. Further detail on the applications of cell decomposition for path finding can be found in [43] [44]. In order to go further on the topic of connectivity graphs, we can mention the *abstraction graphs*. This structure is not a way to discretize space *per se*, but rather a meta-structure that allows to store additional information about the space topology. It allows, for instance, to know if a node is isolated, in a dead end, in a corridor or at a crossing [46] (see fig. 2.7). This technique allows to discard useless research in the graph in a later path search, since all information about further connectivity is stored in the nodes, which can be particularly useful in situations involving closed environments.

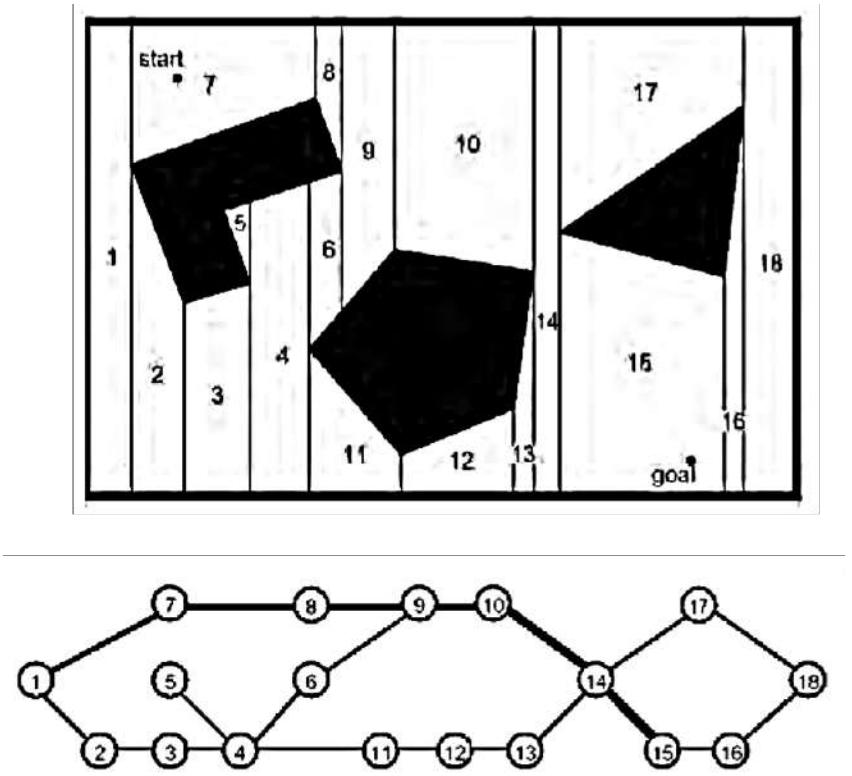


Figure 2.6 – A decomposition with parallel segments and the associated connectivity graph (from [45]).

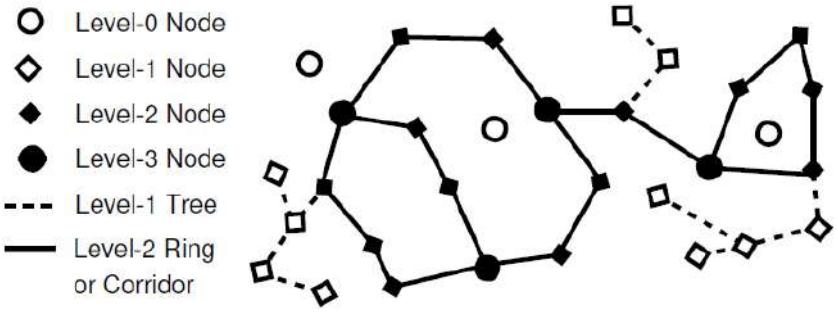


Figure 2.7 – An example of abstract graph in [46].

2.2 Resolution methods for path and trajectory finding problems

The path and trajectory planning problem, as shown in the previous section, can take many shapes, depending on the provided data and the expected results. Depending on the form chosen for the route representation, different types of optimization problems can be considered to tackle such problems. The corner stone in the path and trajectory planning is often to be able to efficiently design one route, as many works in which several routes are to be designed proceed sequentially. Another frequent concern in this topic is the choice of the objective function. In some cases, and more particularly

in many air traffic related works, there are multiple objectives to satisfy at the same time, which are often contradictory (in industry, this is often translated in terms of safety/comfort vs savings/profit ; for another example, the reader can refer to [47]). In these situation, a solution is to introduce the *Pareto Front* of solutions. In order to define the Pareto Front, we say that a solution to a multi-objective optimization problem *dominates* another one when all criteria are better satisfied in the first one. A Pareto front is then defined by the set of solutions that are dominated by no other. In the rest of this document, whenever several criteria are to be optimized at the same time, we will use the term "optimal" to qualify a solution that is Pareto-optimal. In this section, we present some of the methods used to solve the path and trajectory planning problem, gathered in two categories: exact methods, and heuristics or meta-heuristics.

2.2.1 Exact methods

2.2.1.1 Mathematical interpolations

Some methods consist in defining a path as a combination of known basis functions. Finding an optimal path then boils down to finding the optimal coefficients for the combination. Some of these basis functions can be found in [48] and in [49]. In the following paragraphs, we assume that the path to be designed is constrained by a given sequence of points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, through or by which the path must pass.
The most intuitive way to interpolate a given sequence of points is by connecting them with segments. This method is the *piecewise affine interpolation*. In this case each segment $S_i, 1 \leq i \leq n$ is defined as the following function:

$$S_i(x) = a_i(x - x_i) + b_i, x \in [x_i, x_{i+1}] \tag{2.1}$$

where $a_i = \frac{y_{i+1}-y_i}{x_{i+1}-x_i}$ and $b_i = y_i$. An example of piecewise affine interpolation is given in fig. 2.8. The main drawback of this method is that the

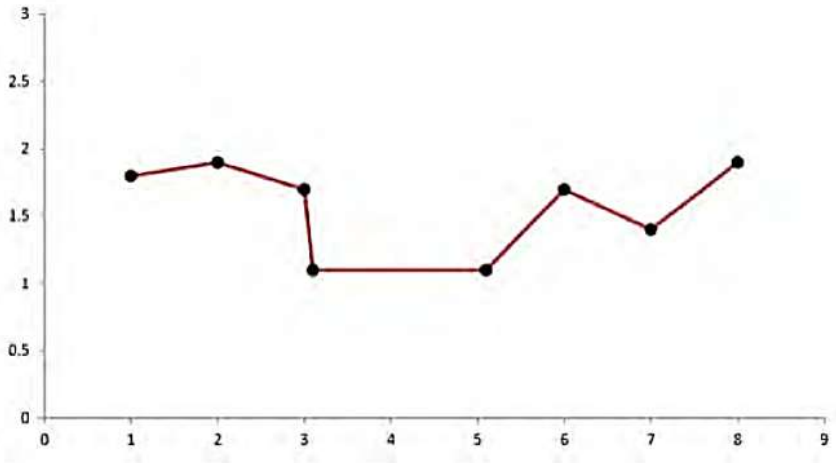
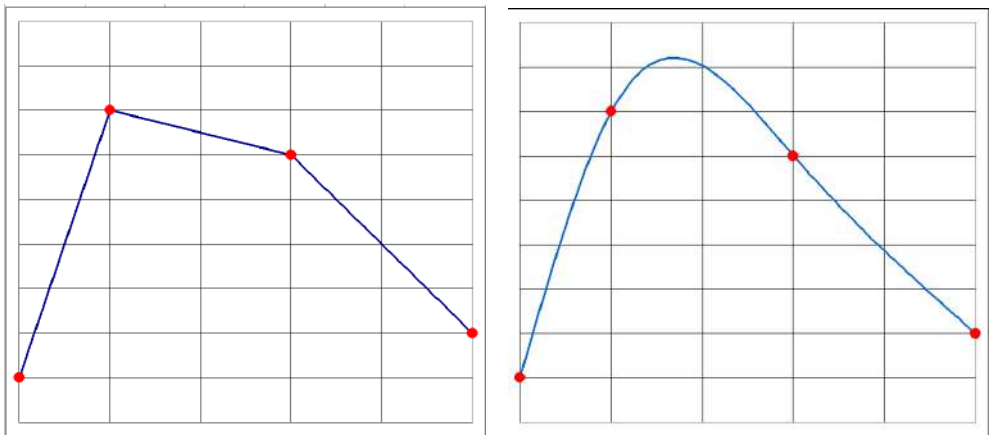


Figure 2.8 – An example of piecewise linear interpolation (from [45]).

path provided is not smooth, its derivative is not continuous. In some cases, like in robotics, this can cause problems of maneuverability. The piecewise linear interpolation belongs to a larger family of interpolations, the *piecewise polynomial interpolations*. In the same fashion as piecewise linear interpolation, this method allows to connect the given points, but this time with polynomials of any order instead of segments. Such curves are also called *splines*. The main advantage of this representation is that the curvature of the result can be constrained, particularly at the control points, making the derivatives of the function continuous. The higher the order of the polynomials, the smoother the curve (see fig.2.9). The *degree* of the spline is the one of the polynomial of highest order. When all polynomials are of the same order, the spline is *uniform*. The particular case of degree 3



(a) The interpolation of 4 points with a spline of degree 1.

(b) The interpolation of 4 points with a spline of degree 3.

Figure 2.9 – Illustration of spline interpolation.

splines is called the *piecewise cubic Hermite interpolation*.
Another way to connect a given sequence of points is to create a unique function instead of a piecewise function. The most famous interpolating function in this domain is the *Lagrange interpolating polynomial* [50]. It is the polynomial of minimal degree that passes through every given point. It is defined as follows:

$$\sum_{i=0}^{n} y_i \prod_{\substack{0 \le j \le n \\ j \neq i}} \frac{x - x_j}{x_i - x_j} \tag{2.2}$$

This definition shows that for every x_i, only one of the sub-products is nonzero. An example of Lagrange interpolation polynomial is given in fig. 2.10. In this figure, every colored curve corresponds to one of the terms of the sum (one x_i). They all evaluate to 0 on all but one x_i. The resulting Lagrange polynomial is represented by the dashed line. The main issue with Lagrange's polynomial is the Runge phenomenon, where the interpolating function's fitness to the real function can actually decrease when the number of interpolation points increases, producing oscillations. Another issue, as for the Bézier curves that will be introduced in the next paragraph, is its dependency on all the points. In this particular case, this can cause a

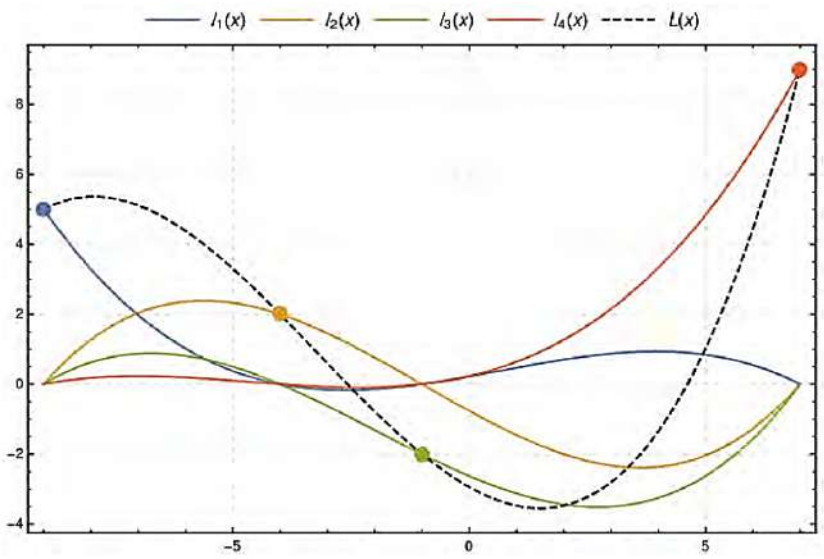


Figure 2.10 – An example of interpolation with Lagrange's polynomial (from [45]).

heavy instability of the solution, depending on the points locations. In some cases, passing through the given points is not required (we then use the term *approximation* instead of *interpolation*), and passing by is sufficient (for instance, this is the case for the waypoints in the RNP procedures in aviation). In these cases, the *Bézier curves* [51] can be applied. By denoting the points $P_i = (x_i, y_i), 0 \leq i \leq n$, the corresponding Bézier curve is defined by:

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} \; P_i, t \in [0,1] \tag{2.3}$$

The curve always starts on P_0 and ends on P_n and usually only passes by the other points, with some exceptions like inflexion points. An example of Bézier curve is given in fig. 2.11. The main drawback of this method,

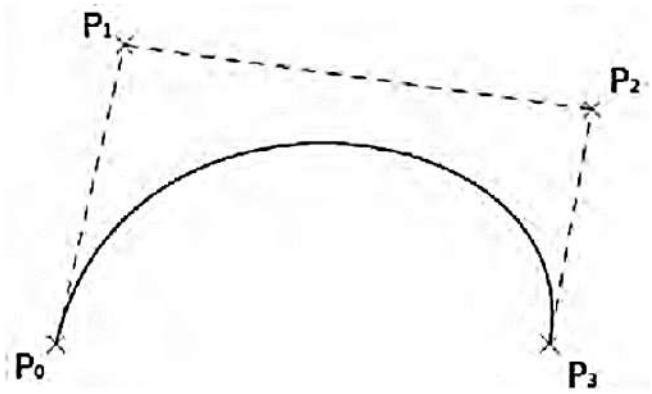


Figure 2.11 – An example of Bézier curve (from [45]).

as for the Lagrange polynomial, is the dependency on all the points. This dependency results in a change of shape for the whole curve when only one

point is changed, but also in an increase in degree when a point is added, which can be difficult to handle when too many points are considered. A way to cope with this problem is to use the equivalent of splines, adapted to the Bézier curves: the *B-splines* (see fig. 2.12).

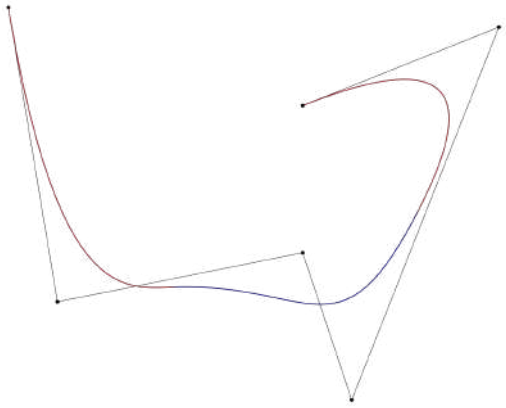


Figure 2.12 – An example of B-spline ([52]).

2.2.1.2 Exact path-finding algorithms

In a path planning problem, the objective to attain can take several forms. It could consist in simply finding a collision-free path, or a path minimizing or maximizing a given criterion. This second objective falls into the category of path optimization, and the most common objective to attain is to find a path of minimum cost. The most famous of these algorithms, and one of the most used as of today, is the *Dijkstra algorithm* [53], that has been widely studied since it came out. It consists in finding the shortest path between two points in a weighted graph that does not contain any *absorbing circuit*, which is a closed path in the graph that has a negative total weight. The basic idea is to iteratively expand the node with the lowest known cost from the start, denoted V_0, by updating the cost of all its adjacent nodes $V_1, \ldots, V_n$. The new cost of a node $V_i, 1 \leq i \leq n$ is the minimum between the current cost of V_i and the sum of the cost of V_0 and the cost to travel from V_0 to V_i. When the node to expand V_0 is the ending node, the shortest path is found. The equivalent to the Dijkstra's algorithm for the single-source multiple shortest paths problem is the *Bellman-Ford* algorithm. First proposed by Alfonso Shimbel in 1955 [54], it was published by Lester Ford Jr. in 1956 [55] and by Richard Bellman in 1958 [56], to whom it owes its name. The main difference with Dijkstra's algorithm is that the Bellman-Ford algorithm computes the shortest distances to *all* vertices in the graph where only the shortest path to one vertex was found in Dijkstra's algorithm. In order to achieve this goal, the method consists in expanding all vertices at each iteration instead of expanding only the one with the lowest cost. This results in an increased time complexity, but also allows to detect absorbing circuits, which is not possible with Dijkstra's algorithm. A more detailed presentation of the Bellman-Ford algorithm is

given in chapter 4.
Another algorithm with a similar functioning is the *A* algorithm* [57]. It works in a similar way to Dijkstra's algorithm by taking into account an estimate of the "cost-to-go" on top of the current cost of a path (see fig. 2.13). This method has been equally, if not more, studied than

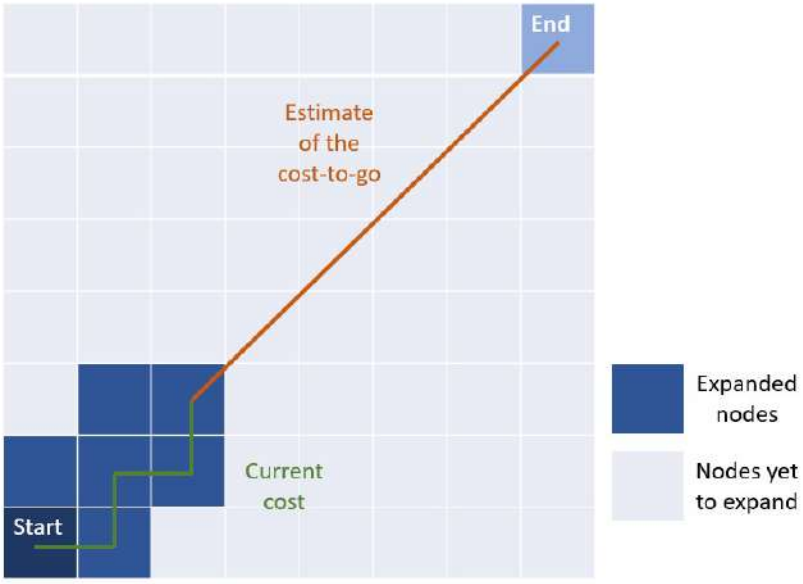


Figure 2.13 – An illustration of the principle of the A* algorithm.

Dijkstra's algorithm. Several variants were developed, so that the algorithm can give a first solution quickly and improve it over time, or so that it can take into account changes in its environment during the execution, for instance [58]. The A* algorithm has been used in a variety of problems, including path and trajectory finding in aeronautical contexts. For instance, in [59], the optimal routes in terminal areas are computed with this method, with considerations of minimum turn angle. Furthermore, it includes a representation of the possible altitudes of an aircraft flying the routes with the means of a cone defined along the 2D path. This idea will be explained more thoroughly later in this chapter. The A* algorithm was also used in the context of trajectory planning under hazardous weather avoidance constraint, for example in [60] and [61]. The A* algorithm has become quite popular in the path finding domain because of its simplicity, and its ability to provide a good result in a short time, which makes it efficient in real-time applications.
A domain in which the path finding in a short time problem is particularly studied is video games [46]. Various techniques are used in this field, most of which rely on triangulated environments. Among them, we will cite here the *Funnel algorithm* [62], designed to find the shortest path in a corridor-like environment. It works by exploring the two sides of the corridor until the straight line connecting the starting point and the current point intersects the boundaries, and then iterates with a new starting point based on this intersection (see fig. 2.14). In video games, environments are modeled with polygons. However, there are applications

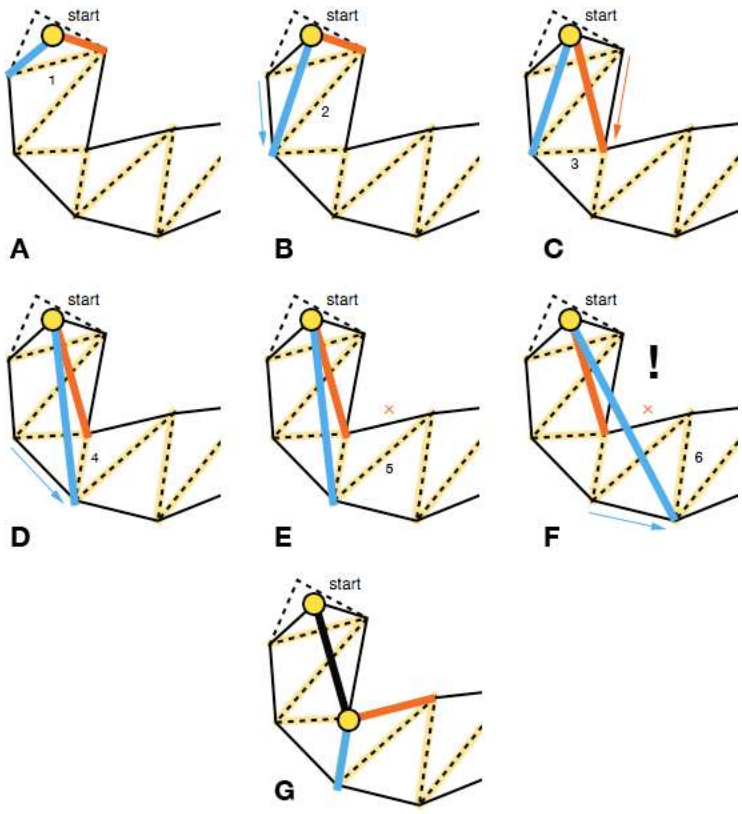


Figure 2.14 – An illustration of the funnel algorithm in [62].

in which the obstacles to avoid have different shapes. In some of these cases, the *Dubins paths* can be used. Their particularity compared to the other methods seen so far is that they consider circular obstacles, and so they don't only rely on segments to construct a path, but on a succession of segments and arcs. It has been proved that Dubins paths are optimal when dealing with circular obstacles [63]. They are still studied as of today, particularly with the aim of integrating them to other path finding methods or with meta-heuristics [64] [65], including in the field of aircraft maneuvering [66][67].
In the line of exact algorithms that tackle combinatorial problems, we will also mention methods like Mixed Integer Linear Programming (used, for instance, in trajectory planning: [68] [69]), where the objective function and the constraint functions are linear and some of the variables can only take integer values. The main drawback of these exact methods is their computation time (which is often exponential with the size of the search space), which makes them unusable on large instances, at least when used only by themselves without hybridization with a heuristic-based approach.
Some works take their inspiration in the processes found in nature, for instance the light propagation phenomenon. In these approaches, in order to simulate the propagation of light in the search space, the *fast marching* method (FMM) can be applied. In principle, the fast marching algorithm in the case of shortest paths search works in a similar way than Dijkstra's

algorithm, and measures how fast a wavefront reaches a given point in space when its starting point and speed function are given, with the possibility that the front doesn't propagate uniformly in space, depending on irregularities or obstacles (see fig. 2.15).

The fast marching method is quite widely used in the domain of wave

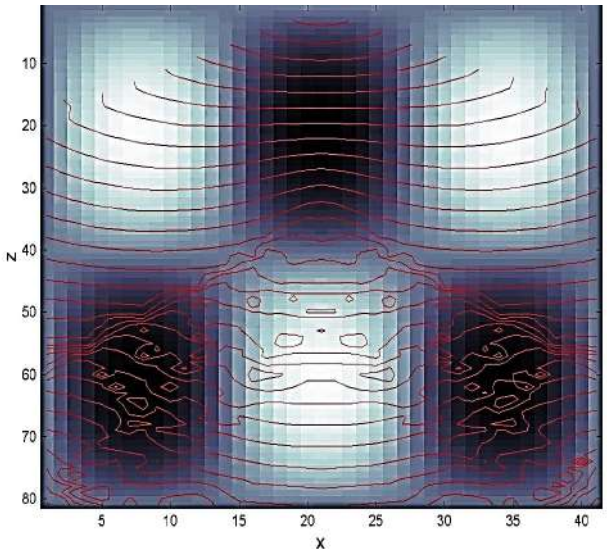


Figure 2.15 – An illustration of wavefront propagation in complex environments in [70].

propagation [71] [72]. This method can be viewed as a particular case of *Optimal Control*, which consists in finding a command for a given dynamical system described by differential equations that is optimal with reference to a given objective function. For instance, it can be used to find the way to pilot an aircraft, given the equations of its dynamics, from a starting to an ending point, that minimizes its fuel consumption. This topic is addressed in [73] and [74], where the problem is modeled in a way that takes into account many operational aspects of air traffic control, such as the various speeds in use. In these works, the aim is to minimize the fuel consumption of aircraft by computing an optimal trajectory while avoiding possible intruders. Moreover, a focus on the use of this method for departure trajectories is provided, with the example of two aircraft departing from Barcelona airport. In previous works, a similar model based on optimal control was used to design departure routes that minimize noise over sensitive areas (see fig. 2.16) by using fuzzy logic, and lexicographic ordering of the objectives and fairness criteria so as to design routes that don't penalize one area for the profit of the others [75] [76] [77]. In these works, the main goal is to minimize the noise disturbance over populated areas. When the noise disturbance has reached a satisfactory level, the authors optimize the design for fuel consumption as an economical criterion, while keeping the trajectory within the acceptable levels of noise. The topic of fuel consumption optimization by optimal control is also addressed in [78], with a less comprehensive model. Another criterion of optimization can be to resolve a maximum number of conflicts between several aircraft.

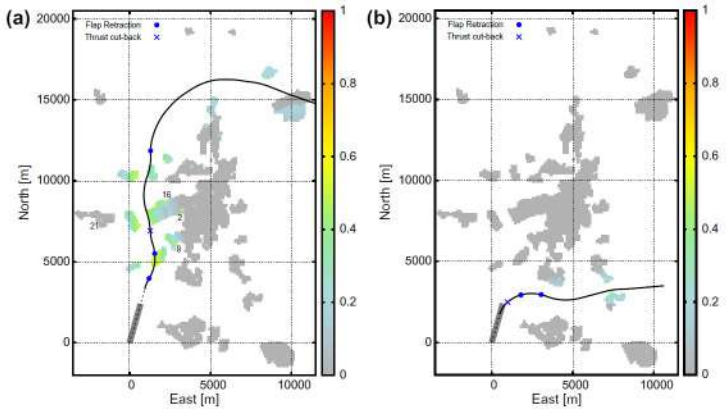


Figure 2.16 – The result of the optimization process in [77] with their associated noise disturbance. The optimal trajectories minimize the maximum noise over each area. The results are shown for two 10am flights, an Airbus A340 (a) and an Airbus A321 (b).

This problem is addressed in [79], with the following works [80] and [81] integrating the managing of the route merging. In this case, it consists in computing both the optimal trajectory for each aircraft and their sequencing upon arrival at their destination. In [82], the authors use an optimal control approach, solved by both direct and indirect methods, in order to optimize the noise reduction around airports. A survey of optimal control methods applied to trajectory planning can be found in [83].

2.2.2 Heuristics and meta-heuristics

The main problem with exact approaches is that they can take too much time to solve the given problem. Their complexity is often exponential with the size of the search space. In order to obtain solutions in a more acceptable time, one can turn towards the use of heuristics and meta-heuristics, that try to estimate as accurately as possible the best possible solution. Some methods mimic natural phenomena in order to compute the shortest path. One method to generate several routes is that of the *fields of potential*. This method was first introduced in 1980 in the domain of robotics [84], and rapidly improved to expand its possibilities, like the coordination of several agents [85], or its application to the design of air routes [86]. The point is to attribute a "power of attraction" to the arrival point(s), and a "power of repulsion" to the obstacles. This generates a map of attraction fields that needs to be traversed in the opposite direction of the attraction field from the arrival point(s) until the starting point(s) to find the optimal path(s) while avoiding the obstacles (see fig. 2.17: a path is constructed with fields of potentials, with the arrival point in green). This method has two main drawbacks: first, it induces oscillations of the trajectory in the vicinity of obstacles, while it should be as smooth as possible. The second drawback of this method is, as other research works, that it remains theoretical in the sense that it does not take into account the current and short-term

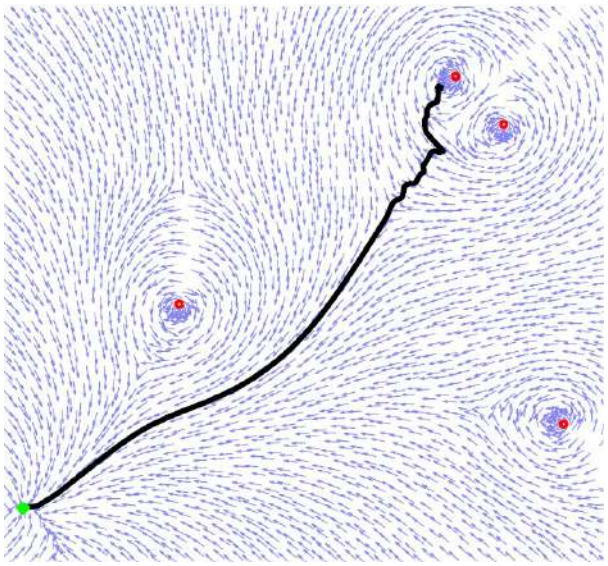


Figure 2.17 – An illustration of the method of the fields of potentials in [86].

operational context of air traffic. Thus, for instance, all paths created with this method merge on the arrival point, which is not manageable from a controller's point of view.
Another method imitating behaviors found in nature that was investigated is the *wavefronts propagation*. It aims at applying *Fermat's principle*, which states that light always follows the fastest possible path. In principle, the search space is assimilated to a propagation medium with a variable refractive index, higher in areas covered by obstacles than in empty areas. This method has been used in numerous fields, from path planning to boundary detection [87] [70], and also applied to the route finding problem for aircraft (but mainly in the en-route context) [88] (see fig. 2.18: a path is designed with the lowest refractive index in blue and the highest in red), with roughly the same problems than with the fields of potential. This work, in order to simulate the light propagation, makes use of a fast marching method.
The previous methods are mostly designed for a specific problem. In order to be able to tackle several types of problem with the same approach, another family of algorithms has been developed, more generic: the *meta-heuristics*. Among them, the *Particle Swarm Optimization* [89] and its derivatives. This meta-heuristic is able to tackle problems of large dimension by finding a global optimum without being dramatically affected by local minima, as can be the case with other algorithms. It works by mimicking the behavior of animal swarms in real life. Each individual moves with respect to three criteria: its velocity, the best point stored in its history and the best point known to the entire swarm. This method allows an efficient exploration of the search space. In an enhanced version, this algorithm allows to tackle multi-criteria problems by using a cooperation/competition system [90]. In this version, there are as many sub-swarms as there are criteria to be optimized. Then, a system of duels allows to choose a representative sub-swarm for each criterion. The cooperation phase then aims at putting

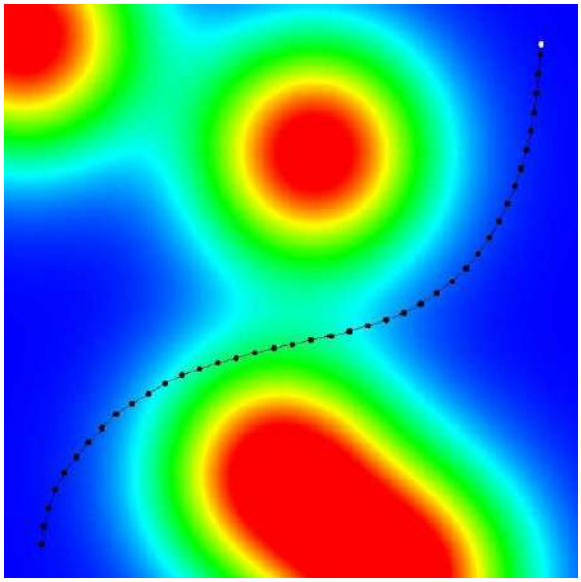
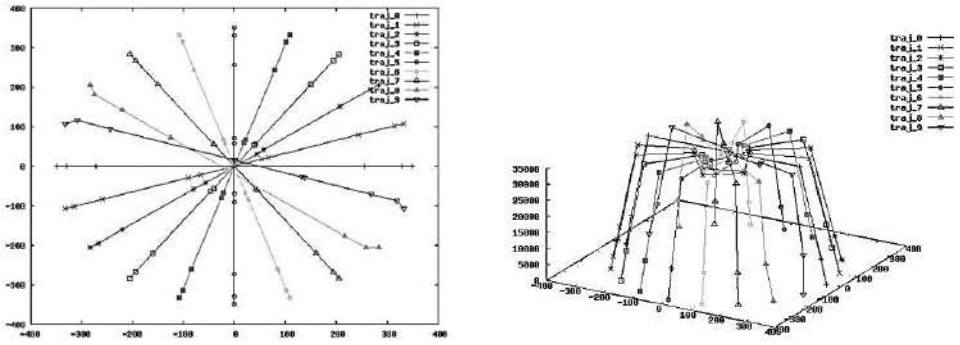


Figure 2.18 – An illustration of the method of light propagation in [88].

together the information collected about the best solutions found by the swarm.
In the family of meta-heuristics, another important type of algorithm is the *Genetic Algorithm* (GA) [91]. It consists in mimicking the process of evolution of living organisms: here again, it involves several individuals. They are iteratively put through a *selection* phase, where the less fit individuals are removed from the group, and then the remaining individuals are paired with one another and produce offsprings. During this process, the offspring inherits each of its characteristics from one parent or the other, and has a low probability of being subject to a *mutation* (usually 0.1 to 1% chance). Note that one pair can generate several offsprings, usually depending on this pair's fitness. This process is designed to build a population that is as fit as possible for the problem under consideration. This method has been applied to the problem of path and trajectory finding in the aeronautical context, for instance in [92] and the following works [93] [94], where aircraft trajectories are designed between pairs of airports, with conflict resolution carried out with the use of Cleared Flight Levels (CFL), allowing for a separation of trajectories in altitude. An example of result found with this method is presented in fig. 2.19
The Genetic Algorithm and its variants (such as the well-known *Non-Sorted Genetic Algorithm* (NSGA-II) [95] for multi-objective problems) can be seen as particular cases of *Evolutionary Algorithms* (EA). In the recent years, another type of EA has been proposed, the *Multiobjective Evolutionary Algorithm based on Decomposition*(MOEA/D) [96], that works by decomposing a multi-objective problem into mono-objective subproblems and solving each one of them independantly in order to find a solution to the original problem. This method has been applied in the domain of SID design in the works of Ho-Huu et al., initially to find SIDs that minimize noise exposure for the inhabitants, as well as fuel consumption for the aircraft [97] [98], and then by integrating to these objectives a third one,



(a) Top view. (b) Side view.

Figure 2.19 – The result of the GA algorithm for 10 paths in [92].

consisting in allocating flights to the routes [99] [100]. Figure 2.20 shows the result obtained for the route design, along with the noise level (green curves) and number of people annoyed by the design. In complex optimization

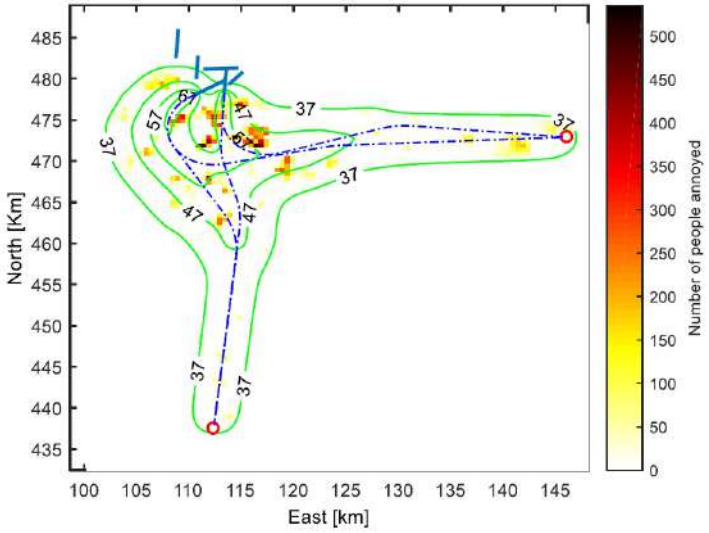


Figure 2.20 – The result of the design of two SIDs along with their noise impact in [100].

problems, it is common to see a solution featuring a meta-heuristic that uses exact methods to solve smaller sub-problems. This is the case, for example, in [101] where several routes are designed sequentially by using a Fast Marching Method (FMM) [102]. In this representation, each previously generated route is viewed as an obstacle for the others, which penalizes the last ones in terms of available space. Therefore, a Simulated Annealing meta-heuristic is added, that tests several possible orders for the route generation. The Simulated Annealing has already been used and proven efficient on problems relative to ATM [103] [104]. Sometimes, the sub-problem itself is solved with a meta-heuristic, resulting in the use of two (or more) meta-heuristics. This is the case, for instance, in [105], where the problem is that of designing a network for coal mining in a complex terrain. A first solution is computed by using an *ant colony* algorithm. This

meta-heuristic aims at mimicking the behavior of an ant colony to find an optimal path. The problem in the scope of the work under consideration is that this method yields a precocious result. In order to solve this problem, the authors integrate the ant colony algorithm into a SA. The possibilities for hybridization are many, each meta-heuristic presenting its own strengths and weaknesses.
Finally, another family of algorithms, the *Probabilistic RoadMap planners (PRM)*, allows to explore the state space with little risk to be caught in a local minimum. It consists in picking random points in the state space (it can take into account more than just a position, for instance an orientation or speed), and in trying to connect these states with one another by means of an admissible path. An admissible path is such that it contains only points in the state space. Therefore, for instance, it cannot pass through an obstacle. Quite many works have been conducted on the subject, although it is still studied as of today, particularly in robotics [106] [107]. In the category of the PRM, a type of algorithms is particularly adapted to the path finding problem in high dimensions: the *Rapidly exploring Random Trees (RRT)* [108]. Their functioning is basically the same as the PRM in the way that it is based on picking random points in the search state and trying to connect them with the previously generated points. Usually, the points are picked one by one and connected to the closest point of the current solution, or the one generating the lowest cost to connect. When the point cannot be connected with the existing solution, it is discarded and another one is drawn. By working this way, a tree structure is created, that eventually reaches the arrival point(s) (see fig. 2.21). Several works have improved this algorithm to be more time-efficient [109] [110], or to allow it to go through more complex obstacles [111] [112] [113]. The strong advantage

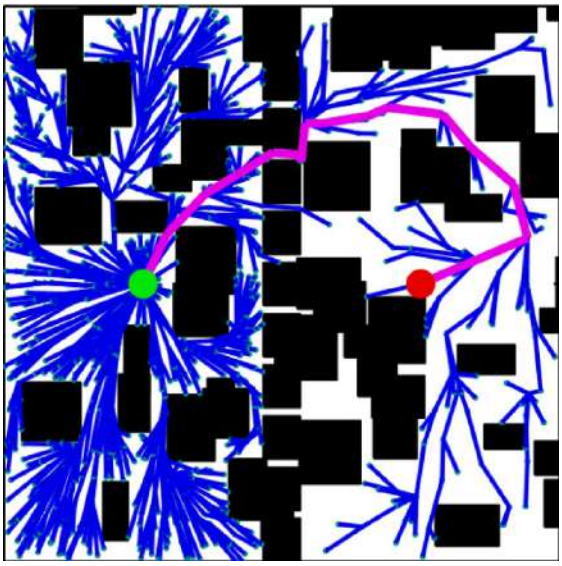


Figure 2.21 – An example of Rapidly exploring Random Tree in [113].

of PRM and more particularly RRT is that they are but lightly affected by the dimension of the state space, where other families of algorithms would see their complexity heavily increased in high dimension search spaces.

2.3 SID and STAR optimization

In comparison with the route and trajectory design problems, the SID and STAR design problem has been less extensively researched. The main reason for this is the large number of constraints that are to be taken into account, and the 3D nature of the problem, which make it difficult to construct a simplified model for the problem. Two of the works that tackle the SID/STAR optimization design are presented below. The first one aims at constructing the STARs in the vicinity of Stockholm Arlanda's airport with an Integer Programming (IP) based method [114]. The second one makes use of the Simulated Annealing and the Branch-and-Bound (B&B) methods to optimize the SIDs and STARs on various scenarios [45]. This thesis relies on these two works from the literature for the tests undergone, in order to compare the method used to the state of the art. These two works focus on different aspects in the design of SIDs and STARs, but both aim at optimizing the procedures.

2.3.1 Automatic Design of Aircraft Arrival Routes with Limited Turning Angle

The first work to be presented is aimed at solving the optimal design of STARs problem in the vicinity of Stockholm Arlanda airport [114] and relies on Integer Programming (IP). It focuses on the operational usability of the output. In order to achieve this goal, the constraints taken into account are:

- no more than two routes merging on the same point;
- two merging points must be far enough from each other;
- no sharp turns;
- obstacle avoidance;
- vertical separation between the SIDs and STARs, with their crossing only authorized at a minimum distance from the runway.

The idea is then to model the surroundings of the runway with a grid, which will serve as an underlying graph for a path search. The grid's cells are designed so that the various routes can merge at a sufficient distance from one another and create a solution that is acceptable for air controllers. Two objectives are taken into account: the individual length of each route, and the *weight* of the solution, which measures the total space that it occupies. An example of solution found with this method is provided by fig. 2.22. The performed tests allowed to find a Pareto front for the solutions, displayed in fig. 2.23. One solution is represented by a set of edges of the grid. In order to further enhance the design, a post-processing technique is applied, which consists in removing any point that is not critical for the maximum turn angle constraint. This process is illustrated in fig. 2.24. The problem of route design with separated merging points is developed by some of the same authors in another work [115], that takes into account the possibilities

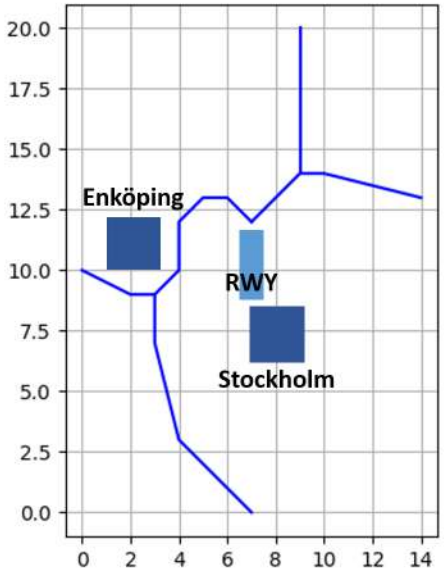


Figure 2.22 – An example of solution in [114].

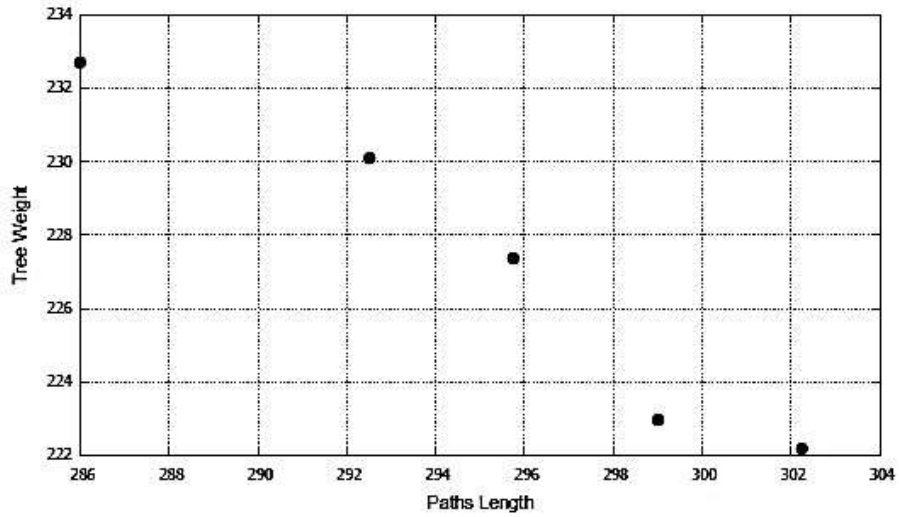


Figure 2.23 – The Pareto front found in [114].

Figure 2.24 – The post-processing technique used in [114].

of RNP, and more particularly the Radius-to-Fix possibility. Therefore, the routes in this work are modeled as successions of segments and arcs of circles.

2.3.2 Optimal Design of SIDs/STARs in Terminal Maneuvering Area

This thesis greatly relies and elaborates on the results found in [45]. Therefore, in this paragraph, we present this work in more detail. As in our work, the aim of the thesis mentioned is to automatically design an optimal set of departure and arrival routes in the vicinity of large airports, in 3D. A route is modeled in two parts:

- the horizontal part, as a Dubins path (a succession of segments and arcs of circles);
- the vertical part, as a cone of altitudes representing all vertical paths that aircraft can take along the corresponding horizontal part.

In [45], obstacles are taken into account. They can be either ground obstacles (mountains, buildings...) or air obstacles (like forbidden zones), with a nonzero minimum altitude. These obstacles are represented as cylinders, made up of:

- a base, given as a circle, with its center and radius;
- a minimum altitude;
- a maximum altitude.

A representation of an obstacle is given in fig. 2.25. In her work, the

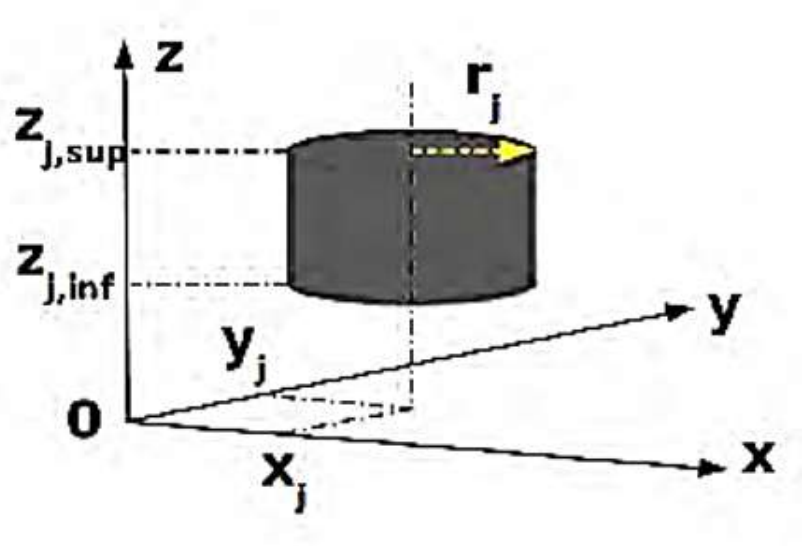


Figure 2.25 – The modeling of an obstacle in [45].

author considers that two obstacles whose projections on the horizontal plane completely overlap (one base circle is totally included in the other) are one and the same. This prevents from letting the possibility to pass between the two obstacles. Conversely, there are obstacles, such as mountains, that cannot be represented by only one cylinder, as the approximation wouldn't make any sense. In these cases, the obstacle is broken down into several sub-obstacles that are more fitted for the problem (see fig. 2.26).

The chosen shape for the obstacles in [45] offers four possibilities to bypass

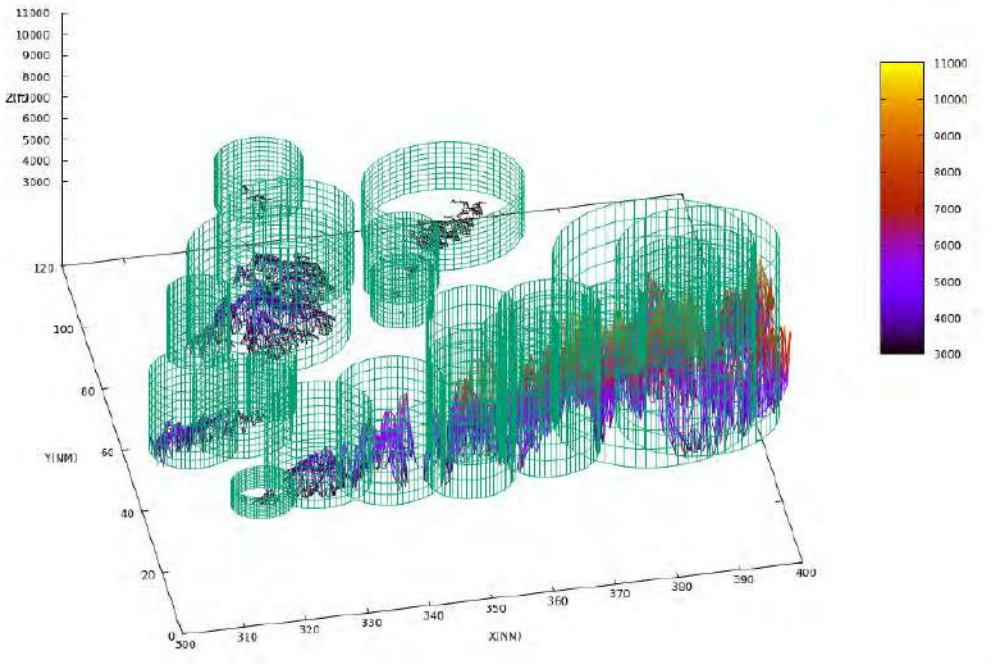


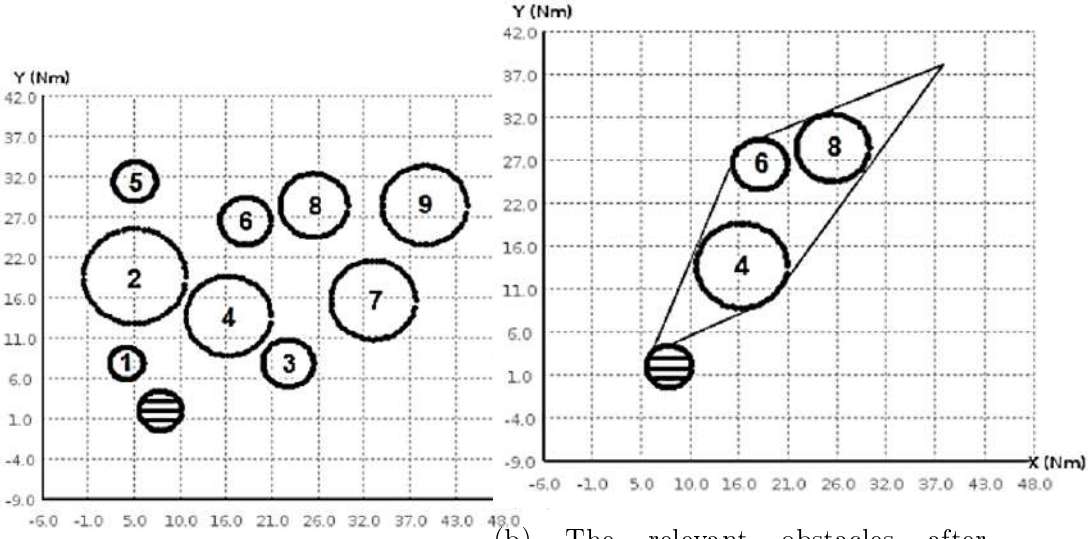
Figure 2.26 – Modeling of the obstacles in the Zurich airport region in [45].

them:

- clockwise;
- counter-clockwise;
- by passing underneath;
- by passing above.

These possibilities are at the core of the method chosen to build the optimal paths. The author solved the optimal path finding subproblem for one route with a Branch-and-Bound (B&B) algorithm. In this case, the branching strategies are the way to bypass obstacles. Therefore, all paths designed with this method will automatically be collision-free. However, this also implies that the time required to run the algorithm is directly linked to the number of obstacles to be taken into account. In order to reduce this number, a convex hull filter is applied before launching the algorithm (see fig. 2.27). Applying the filter allows to greatly reduce the number of obstacles to be accounted for in some cases, allowing the algorithm to run faster. In this work, the author also takes into account the orientation of the runway. This is done by creating an artificial obstacle near the starting point of the route, and imposing the way to bypass it. By doing so, the initial orientation of the route is constrained (see fig. 2.28: the fictitious obstacle is striped).

The design of several routes is done in two ways. In the first one, the author applies a 1 vs all strategy. This consists in designing the routes in a given arbitrary order. Each designed route serves as an obstacle for the next ones. For each route, the design is firstly done by considering only



(a) All starting obstacles.

(b) The relevant obstacles after applying the convex hull filter.

Figure 2.27 – The effect of the convex hull filter in [45].

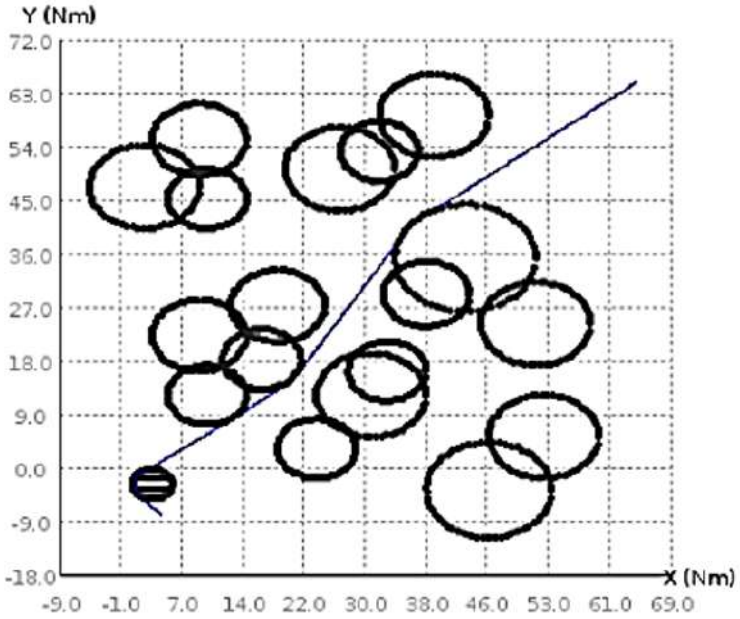


Figure 2.28 – The design of one route in [45].

the original obstacles. Then, if the newly designed route is in conflict with a previously generated route, a new fictitious obstacle is added, covering the area of conflict, and the last route is designed again, by taking the new obstacle into account. This allows the routes to be separated in the vertical plane (see fig. 2.29: the route γ_3 takes two detours to avoid γ_1 by passing above it and γ_2 by passing below it). The conflict detection is carried out by sampling the paths with a constant step and putting the resulting points into a grid. Each cell of the grid is then analyzed: if it contains a point, all neighboring cells are checked for points from other paths. When another

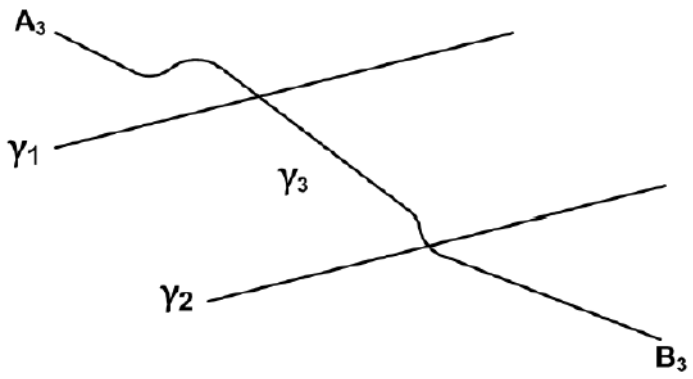


Figure 2.29 – The conflict resolution strategy in [45].

point is found, the vertical separation of the routes is evaluated. There is a conflict whenever the separation is too small (this method is explained in more details in chapter 3).
This method of designing the routes imposes to choose beforehand an order for the design. Therefore, another way to design the routes was tested, using the Simulated Annealing meta-heuristic. In this method, all routes are generated regardless of the conflicts between routes. When all routes have been generated, a neighboring solution is designed by adding fictitious obstacles for all routes on each area of conflict and choosing an avoidance strategy for these obstacles. The algorithm iterates by choosing various avoidance strategies.
Both methods were tested on a certain number of instances. Among them, the cases of Paris Charles-de-Gaulle and Zurich, which will be taken as references for the tests in this thesis. As stated before, the objective of the work from [45] is to produce the shortest possible routes. Incidentally, all routes merge on the same point, which is the runway threshold. Such a configuration is not manageable for a controller, as the workload induced by the situation is too important. This constitutes the main drawback of the method.

2.4 Conclusion

Although the topic of path and trajectory planning has been quite extensively researched, the specific problem of SID/STAR design is still to be solved efficiently in the current state of air traffic operations for large airports. The main difficulty of such a design is the number and complexity of the constraints, which doesn't allow for the exclusive use of exact methods, that would take too long to yield a satisfactory result. The works that have already been done on the subject present two major drawbacks:

- The method does not take into account the current state of air traffic operations. As a result, the solutions found are often impossible to use in real-life scenarios although they are compliant with the mathematical objective that was assigned.

- The method is not suited for large instances. Some approaches work fine on situations with few routes to design or with a narrow search space, but cannot give a satisfactory output on larger instances in an acceptable time.

The topic raises the problem of multiple optimal path finding in a 3D environment, for which the literature does not yet provide fully satisfactory solution. This work provides a method that allows to take into account many operational constraints (management of the merging points between the routes, terrain avoidance, forbidden zones, limited turns...) and several objectives (noise reduction, distance flown...) at the same time, including on large instances. The topic requires a specific mathematical model in order to be addressed in a rigorous way. In the next chapter, we present the mathematical modeling that was chosen for the SID/STAR optimization problem, including the variables, the constraints and the objective function.

Chapter 3

Problem modeling

This chapter depicts the problem of SID/STAR design as an optimization problem. We describe the mathematical modeling that was used to establish its formulation. This chapter is divided into three sections. First, we present the data that we consider known in advance, and provided as input for the resolution method. In the second section, we explain how the TMA is represented, as well as the shape that was chosen for the routes to be designed. In the last section, we formulate the problem as an optimization problem. We express the constraints and the objective function.

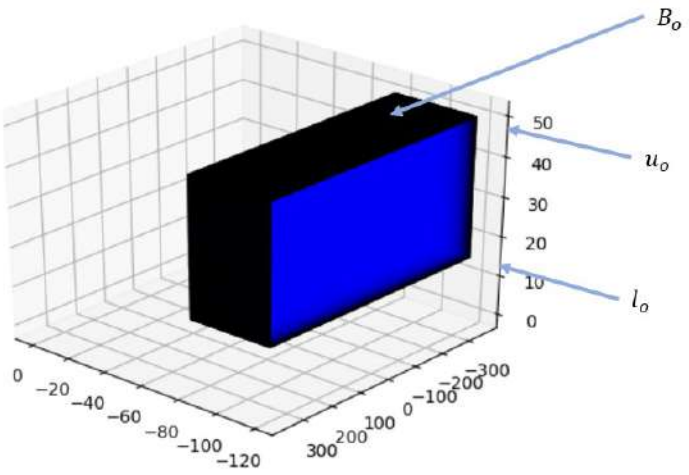
3.1 Input data

In order to be able to design SIDs and STARs, some prior data is required. The first element to know is the departure and arrival points of each route to be designed. Then, the highest points of the terrain in the area surrounding the airport(s) are considered. They are the critical elements for route design, as they define the lower boundary of the search space. The shape of the routes depends on their height and positioning. Once these elements are identified, the designers can begin to create the routes, usually by starting with the legs that are closest to the runway, and expanding towards the boundary of the TMA. Here, in order to compute the routes in the TMA, we assume that various parameters are known beforehand. We list them in this section. They are separated into two categories: the elements relative to the environment, and those relative to the aircraft and operations.
The parameters relative to the environment are:

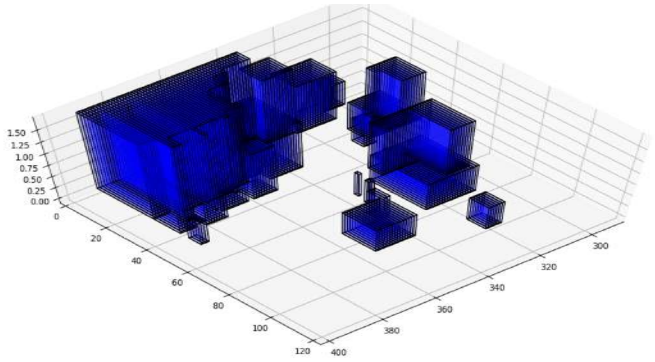
- the position and altitude of the starting point for the SIDs of every runway to take into account, as well as the position and altitude of the ending point for the STARs (for instance the Initial Approach Fix) of every runway to take into account;
- the orientation used for each runway;
- the exit point of each SID to the en-route sector, associated to a range of admissible altitudes, as well as the entry point of each STAR into the TMA, associated to a range of possible altitudes $P^1, ...P^{N_P}$ where N_P is the total number of such points;
- the expected traffic flow on each route to be designed $F^i, i \in \{1, \dots N_P\}$;
- the elevation of the terrain beneath the TMA;
- the set of ground (such as mountains, buildings) or air (such as military zones) obstacles $\mathscr{O}$. An obstacle $o \in \mathscr{O}$ is given as $o = (B_o, l_o, u_o)$ where B_o is the base polygon, l_o and u_o are respectively the lower and higher altitudes of the obstacle (see fig. 3.1);
- the set of cities T underneath the TMA. Each city $\tau \in T$ is given as 2D polygons, as well as their population density: $T = \{\tau = (B_\tau, \eta_\tau)\}$ where B_τ is a 2D polygon, and $\eta_\tau : B_\tau \to \mathbb{R}^+$ is the density function that gives the population density at a given point in the city.

The parameters relative to the operational context are:

- the minimum and maximum climb and descent slopes $\alpha_{\text{min,climb}}$, $\alpha_{\text{max,climb}}$, $\alpha_{\text{min,descent}}$, $\alpha_{\text{max,descent}}$. To simplify the reading, we suppose that $\alpha_{\text{min,climb}} = \alpha_{\text{min,descent}} = \alpha_{\text{min}}$ and $\alpha_{\text{max,climb}} = \alpha_{\text{max,descent}} = \alpha_{\text{max}}$ without loss of generality;
- the maximum admissible turn angle θ_{max};
- the maximum authorized number of level flights n^{LF}_{max};
- the minimum and maximum authorized length of a level flight l^{LF}_{min}, l^{LF}_{max} respectively;
- the minimum horizontal distance to keep with obstacles or other routes d_h;
- the minimum vertical distance to keep with obstacles or other routes d_v;
- the minimum horizontal distance between two merging points d_{m}.



(a) The representation of an obstacle.



(b) Various obstacles.

Figure 3.1 – The modeling of the obstacles.

3.2 Graph construction and route representation

In this section, we describe the construction of the support for the design of the routes and the way we chose to represent them. The approach consists in discretizing the TMA into a suitable graph in which the routes will be searched. We first present the way in which the graph is built, then, in 3.2.2, the way in which the routes are represented based on this graph. The graph gives their horizontal shape to the routes. The vertical part is added once the horizontal path is designed.

3.2.1 TMA representation and route network construction

We chose to turn towards a graph structure to design the routes (see fig. 3.2). More precisely, one graph structure is designed for each runway orientation that is to be taken into account (see fig. 3.3). The vertices are constructed as follows :

- The starting point of the route (which is usually located near the

runway's threshold, in its alignment) is called the *center*, and is denoted C. It represents the first waypoint at which an aircraft will be authorized to maneuver.

- We construct concentric *layers*, centered on C. The layers can be the borders of any family of increasing convex sets (for instance: circles, or squares). They are denoted by the set $\mathscr{L} = \{\mathscr{L}_i, i \in \{1, \dots N_{\mathscr{L}}\}\}$ where $N_{\mathscr{L}}$ is the total number of layers. We consider that C is itself a layer. The layers are constructed so that $\mathscr{L}_{N_{\mathscr{L}}}$ passes through the exit point of the SID.

- Each layer is sampled to create a set of points. We denote $V_i = \{v_{i,j}, j \in \{1, \dots N_{\mathscr{L}_i}\}\}$ the set of all points yielded by the sampling of $\mathscr{L}_i$. In this notation, $v_{i,j}$ is the j^{th} point on $\mathscr{L}_i$ and $N_{\mathscr{L}_i}$ is the number of points contained on $\mathscr{L}_i$. In order to simplify the reading, and without any loss of generality, we will consider in the rest of the document that: $\forall i \in \{2, \dots N_{\mathscr{L}}\}, N_{\mathscr{L}_i} = N$ with N a constant.

This process provides a set $V = \cup_{i \in \{1, \dots N_{\mathscr{L}}\}} V_i$ of vertices. The next step consists in connecting these vertices together by edges in order to obtain a graph. This is done as follows (see Algorithm 1):

- We denote $e^i_{j,k}$ the segment that connects $v_{i,j}$ and $v_{i+1,k}$.

- For a given vertex $v_{2,j}$ located on the second layer, we consider the segment $e^1_{1,j}$ (lines 6-9 for the general case).

- The angle formed by the orientation of the runway and $e^1_{1,j}$ is evaluated (lines 14-15). If it is not greater than $\theta_{\max}$, $e^1_{1,j}$ is added to the set of edges E (lines 16-17).

- This process is done for every point of $\mathscr{L}_2$.

- When all points of $\mathscr{L}_2$ have been considered, the layer $\mathscr{L}_3$ is processed. This is done by repeating the same process, but instead of checking the angle with the initial direction, we check the angle with every edge between $\mathscr{L}_1$ and $\mathscr{L}_2$ (lines 11-13). We then do the same for all subsequent layers.

This process provides an oriented graph $G = (V, E)$, in which it is possible to perform path searches (see fig. 3.2). By construction, a graph is associated to one runway threshold with its initial direction. Therefore, when several thresholds are to be taken into account, one graph is created for each threshold. We obtain a configuration illustrated in fig. 3.3.

3.2.2 The route representation

Based on the graph constructed in the previous paragraph, we are able to define a route. It is composed of two parts. The *horizontal profile* is defined as a succession $(e^1_{i_1,i_2}, e^2_{i_2,i_3} \dots e^{N_{\mathscr{L}}-1}_{i_{N_{\mathscr{L}}-1},i_{N_{\mathscr{L}}}})$ of elements of E (see fig. 3.4). In

Algorithm 1 Construction of one search graph.

Require: $center = (x_c, y_c)$, $\theta_{\max}$ the maximum turn angle, *vertices* a double-entry array containing all vertices, such that $vertices[i][j] = v_{i,j}$, $initialDirection = (x_i, y_i)$ a vector indicating the direction of the runway

Initialization: Let $E := \emptyset$, $previousDirection := initialDirection$, *expectedDirection* a null vector, $currentPoint = center$, *nextPoint* a null vertex, *predecessors* an empty double-entry array indexed by the vertices, such that $predecessors[v_{i,j}]$ contains the predecessors of $v_{i,j}$ as vertices
Set $predecessors[\text{center}] = \overrightarrow{(initialDirection, center)}$
for i from 1 to $N_{\mathscr{L}} - 1$ **do**
    Set $N' = 1$ if $i = 1$, N otherwise
    for j from 1 to N' **do**
        $currentPoint = vertices[i][j]$
        for k from 1 to N **do**
            Set $nextPoint = vertices[i+1][k]$
            Set $expectedDirection = \overrightarrow{(currentPoint, nextPoint)}$
            Set $N_p = predecessors[v_{i,j}]$.length
            for l from 1 to N_p **do**
                $previousPoint = predecessors[currentPoint][l]$
                $previousDirection = \overrightarrow{(previousPoint, currentPoint)}$
                Set $\theta = \text{angle}(previousDirection, expectedDirection)$
                if $\theta \leq \theta_{\max}$ **then**
                    E.add($expectedDirection$)
                    $predecessors[nextPoint]$.add($currentPoint$)
                end if
            end for
        end for
    end for
end for
return E

the rest of the document, we will denote the horizontal path by several possible means, depending on the context:

As a function	$\gamma_h : [0,1] \to \mathbb{R}^2$, or
As a succession of edges	$(e^i_{j,k})_{1 \leq i \leq N_{\mathscr{L}}-1} \in E^{N_{\mathscr{L}}-1}$

In the two cases, the horizontal profile is referred to as γ_h. In the second notation, an edge $e^i_{j,k}$ belonging to the horizontal profile can be referred to as $\gamma_h[i]$. Moreover, we denote:

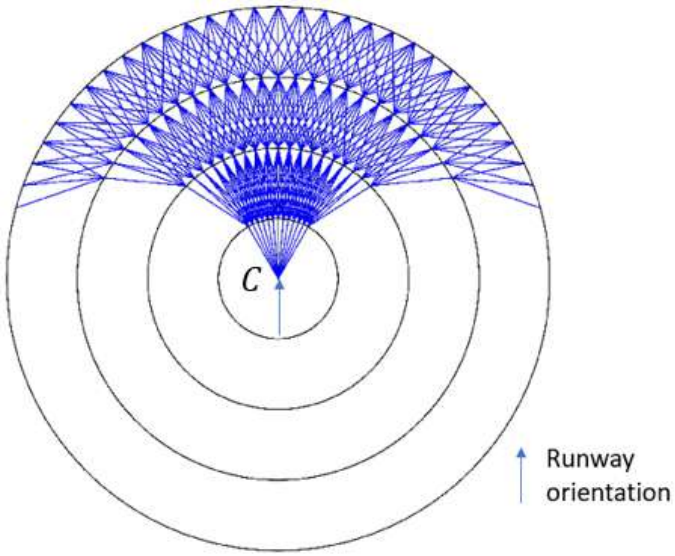


Figure 3.2 – An example of discretization and graph construction (Five layers, one vertex every 5°, $\theta_{max} = 30°$).

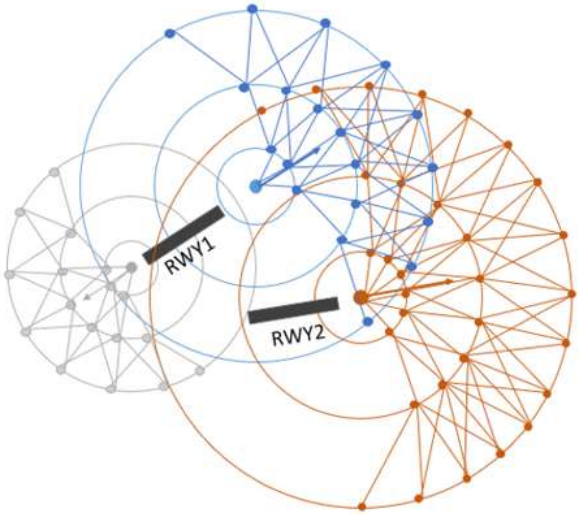


Figure 3.3 – An illustration of graphs for three runway directions.

- $l(\gamma_h) := \int_0^1 \|\gamma_h'(s)\|\, ds$ the total length of γ_h;
- $d(t) = \int_0^t \|\gamma_h'(s)\|\, ds$ the *curvilinear abscissa* at t of γ_h, which is the distance flown along γ_h from the center to t with $t \in [0, 1]$;
- $\gamma_h[l_i, l_j]$ the portion of the horizontal profile that starts at layer $\mathscr{L}_i$ and ends at layer $\mathscr{L}_j$;
- The family $0 = \tau_1 < \tau_2 < ... < \tau_{N_{\mathscr{L}}} = 1$, such that $\gamma_h([\tau_m, \tau_n]) = \gamma_h[m, n]$. This family allows us to describe a continuous portion of a horizontal profile, by using the representation of γ_h as a function.

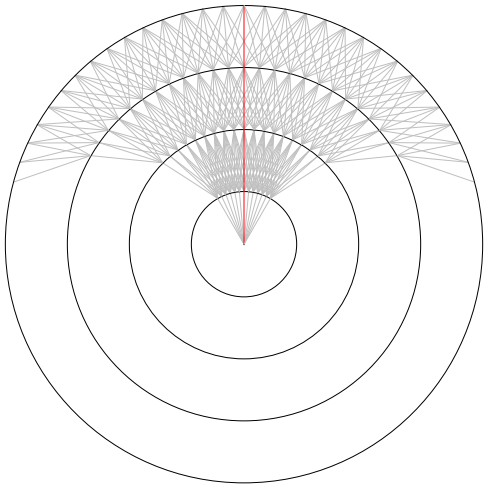
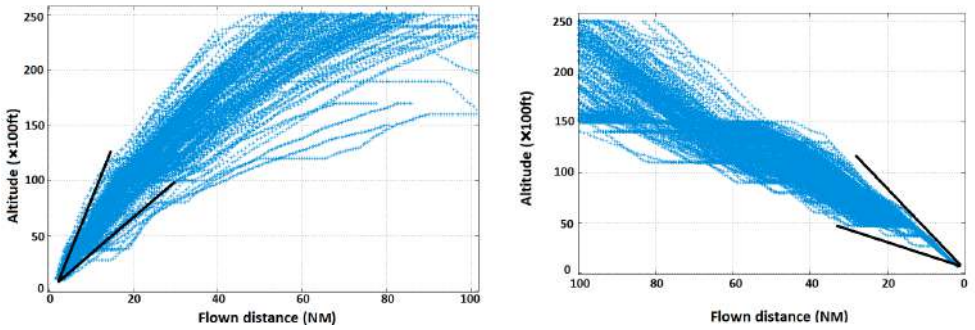


Figure 3.4 – An example of horizontal profile in the graph.

The second part of a route is the *vertical profile*. It is represented by a function $\gamma_v : [0, l(\gamma_h)] \to \mathbb{R}^{+2}$ that gives, for a given curvilinear abscissa, the minimum and maximum altitudes at which an aircraft will be able to fly. In this work, we suppose that the vertical profile is composed of a continuous climb, with possible level flights, in a limited number. This choice has been made so as to take advantage of the CCO concept and its possibilities. It was also motivated by the observation of the take-off and landing profiles at Paris Charles-de-Gaulle airport (see fig. 3.5). In this work a *level flight* is



(a) Take-off profiles. (b) Landing profiles.

Figure 3.5 – Take-off and landing profiles in Paris CDG airport [45].

considered as one or several consecutive edges of a horizontal profile on which the maximum (or minimum, in the case of a descent) altitude is constrained. Hence, given a minimum and a maximum climb slopes $\alpha_{\min}$ and $\alpha_{\max}$, the vertical profile is entirely defined by the family $(z_1, ..., z_{N_{\mathscr{L}}-1}) \in \{0, 1\}^{N_{\mathscr{L}}-1}$ where z_i indicates the presence (1) or absence (0) of a level flight between the layers i and $i + 1$, i.e. on the arc $\gamma_h[i]$. It is then possible to build the function given previously, as explained by Algorithm 2. In essence, this

algorithm builds two functions:

$$\begin{array}{rccl} \underline{z}, \overline{z}: & [0, l(\gamma_h)] & \to & \mathbb{R} \\ & s & \mapsto & \text{the minimum and maximum possible altitudes of an aircraft} \\ & & & \text{at flown distance } s \text{ from the center} \end{array}$$

The functions $\underline{z}, \overline{z}$ are continuous piecewise linear and can be characterized by their values at curvilinear abscissa of the intersection between the route and the layers. By abuse of notation, the values corresponding to layer i are

---

Algorithm 2 Construction of $\underline{z}, \overline{z}$. They are computed layer by layer, by increasing each time the altitude by the distance flown multiplied by the slope (lines 10-15), or by constraining the maximum altitude and updating the minimum altitude accordingly when there is a level flight (lines 6-9).

Require: $HorizontalPath = \gamma_h$, $VerticalProfile = \gamma_v$ in the binary representation, an initial altitude Alt_{init}, a minimum slope $Slope_{min}$, a maximum slope $Slope_{max}$

1: Initialization: Let $\underline{z} = (Alt_{init}, 0, ...0)$ and $\overline{z} = (Alt_{init}, 0..., 0)$ two arrays of length $N_{\mathscr{L}}$, $Alt_{max}^{previous} = Alt_{init}$, $Alt_{min}^{previous} = Alt_{init}$

2: **for** i from 1 to $N_{\mathscr{L}} - 1$ **do**

3: Let $(v_j^i, v_k^{i+1}) = HorizontalPath[i]$ and $Alt_i = VerticalProfile[i]$

4: Let $d = d_{2D}(v_j^i, v_k^{i+1})$

5: Let $Alt_{min} = Alt_{min}^{previous} + Slope_{min}\frac{d}{100}$ and $Alt_{max} = Alt_{max}^{previous} + Slope_{max}\frac{d}{100}$

6: **if** Alt_i is true **then**

7: $\underline{z}[i+1] \leftarrow \min(Alt_{max}^{previous}, Alt_{min})$

8: $\overline{z}[i+1] \leftarrow Alt_{max}^{previous}$

9: $Alt_{min}^{previous} = \min(Alt_{max}^{previous}, Alt_{min})$

10: **else**

11: $\underline{z}[i+1] \leftarrow Alt_{min}$

12: $\overline{z}[i+1] \leftarrow Alt_{max}$

13: $Alt_{max}^{previous} = Alt_{max}$

14: $Alt_{min}^{previous} = Alt_{min}$

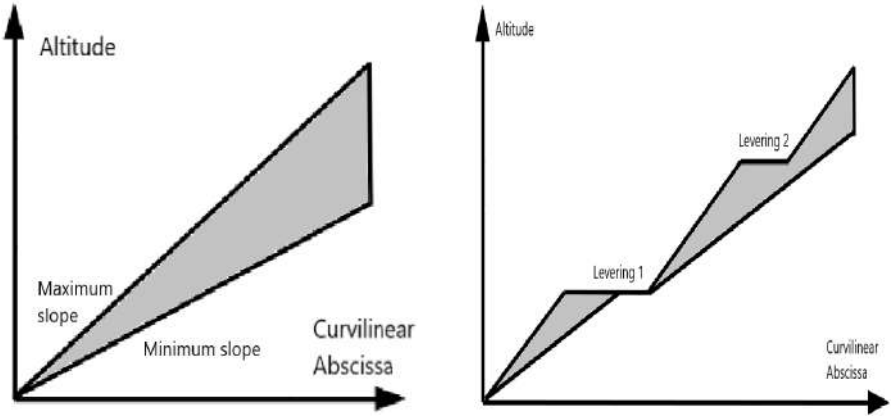
15: **end if**

16: **end for**

17: **return** $(\underline{z}, \overline{z})$

---

denoted by $\underline{z}[i], \overline{z}[i]$. Figure 3.6 gives an example of the construction of the functions $\underline{z}, \overline{z}$, in which four layers are represented (two normal climbs and two level flights in the picture on the right). In the rest of this document, γ_v



(a) A vertical profile with no level flight.

(b) A vertical profile with two level flights.

Figure 3.6 – Illustration of the vertical profile z_γ.

will refer to either of the three formulations, depending on the context. We adopt the same notations as with the horizontal profile: $\gamma_v[i]$ and $\gamma_v[l_1, l_2]$.
A route is the association of a horizontal profile and a vertical profile.
The aim of our work is to be able to design several routes for large airports. In order to do so, we denote by X^i the element X for the i^{th} route to be designed, with $i \in [1, N_P]$. For instance, we will denote γ_h^3 the horizontal profile of route 3.

3.3 Optimization problem formulation

In the previous section, we presented the underlying structure of the routes and the way to design them. Based on these representations, we are able to formulate the problem of the SID/STAR design as an optimization problem. In this section, we introduce the variables of the problem, then its constraints, and the objective function.

3.3.1 Decision variables

The problem consists in designing a set of routes in an optimal way. Therefore, the two variables at stake are:

- the horizontal profiles of the routes γ_h^i;
- the vertical profiles of the routes γ_v^i.

Since several routes are to be designed, we must also be able to define the connection points between them, as we want these connections to be separated from one another. Therefore, an aspect in the choice of the horizontal profile must be emphasized: the merging points of the routes. We define the *merging layer* of two routes i and j by

$$\mathscr{L}_{ij} = \max \left\{ l \in \{1, ..., N_{\mathscr{L}}\} \;\middle|\; \gamma_h^i[1, l] = \gamma_h^j[1, l] \right\}$$

Note that, by construction, $\gamma_v^i[1, l_{ij}] = \gamma_v^j[1, l_{ij}]$. Also, l_{ij} always exists whenever routes i and j belong to the same graph and can be 1. In this case, the routes i and j only have the center in common. The *merge point* is then introduced as the common node between γ_h^i and γ_h^j located on layer $\mathscr{L}_{ij}$. See fig. 3.7 for an illustration of a merge point.

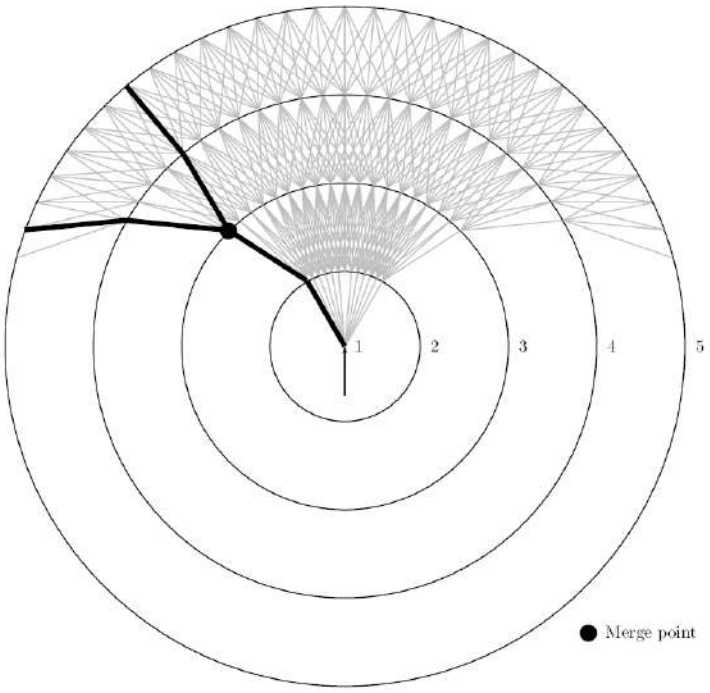


Figure 3.7 – An illustration of merge point.

3.3.2 Constraints

In order for our work to yield relevant results, we must take some constraints into account. The problem of SID/STAR design is very complex due to the number, and sometimes the complexity, of these constraints. They arise from both the operational requirements of safety and operability, which are relative to the shape of the routes, and the environment in which the routes are designed, that constrains the search space. We list these constraints in the following paragraphs.

3.3.2.1 Obstacle avoidance constraint

Avoiding obstacles is the first, and sometimes the most difficult aspect to take into account when designing SIDs and STARs. Here, we refer to ground obstacles, such as buildings, forests or terrain. For safety measures, it is not sufficient to impose that the routes do not cross obstacles. We must provide an additional margin around the route, both horizontally and vertically, to ensure that aircraft will be able to avoid the obstacles. The way to create this margin is quite complex, and depends on the type of procedure. In this work, we chose to take this constraint into account by adding a

constant margin horizontally and vertically to the route. The reader can refer to [116][117][118] for more details on the construction of protection areas. We denote $o = (B_o, l_o, u_o) \in \mathscr{O}$ an obstacle, given as a base polygon in 2D (B_o) and a lower and higher altitude (l_o and u_o). Thus the obstacles are modeled as cylinders. The constraint is expressed as follows:

$$\forall o \in \mathscr{O}, \forall i \in \{1, ..., N_P\}, \forall t \in [0,1], d(\gamma_h^i(t), B_o) \geq d_h \text{ or } \max(\underline{z}^i(d(t)), l_o) - \min(\overline{z}^i(d(t)), u_o) \geq d_v \tag{3.1}$$

In this equation, d is the euclidean distance between two objects (i.e. the minimum distance between two points, one on the first object and the other on the second) and d_h and d_v are respectively the minimum horizontal and vertical distances to keep with an obstacle.

3.3.2.2 Limited turn constraint

Since aircraft cannot maneuver in any direction instantly, the route design must take a maximum turn angle into account:

$$\forall n \in \{1, ...N_P\}, \forall e_{j,k}^{i-1} = (v_j^{i-1}, v_k^i), e_{k,l}^i = (v_k^i, v_l^{i+1}) \in \mathscr{R}^n, | \widehat{v_j^{i-1} v_k^i v_l^{i+1}} | \leq \theta_{\max} \tag{3.2}$$

This equation ensures that two consecutive arcs in the horizontal profile form an angle lower than a maximum angle, with $\widehat{v_j^{i-1} v_k^i v_l^{i+1}}$ being the angle formed by the succession of vertices v_j^{i-1}, v_k^i and v_l^{i+1}. Figure 3.8 illustrates this constraint: it shows a path that violates it by including too sharp turns. This figure also illustrate the fact that even by taking into account the maximum turn angle in the graph's construction, some paths can still violate the constraint. This is due to the fact that the graph is essentially built by taking into account all possibilities of paths and putting all of them together. However, when designing a single path, the possibilities to go from a layer $\mathscr{L}_i$ to a layer $\mathscr{L}_{i+1}$ are constrained by the choice that was made to go from the layer $\mathscr{L}_{i-1}$ to the layer $\mathscr{L}_i$.

3.3.2.3 Route separation constraint

As several routes are to be designed, we must ensure that they do not intersect each other. This constraint is similar to the obstacle avoidance constraint, except that we require an extra margin between the routes in order to avoid *airprox*, the situation when two aircraft are too close to each other, as much as possible. However, this condition cannot be met whenever the two routes at stake merge together. In this case, we impose that the angle between them at their merge point be greater than a minimum angle, so that the route are separated as soon as possible. The constraint is expressed as follows:

$$\forall i, j \in \{1, ..., N_P\}, j \neq i, \forall t \in \left[\tau_{l_{ij}}^i, 1\right], \forall s \in \left[\tau_{l_{ij}}^j, 1\right], \left[\begin{array}{l} d(\gamma_h^i[l_{ij}+1, N](s) - \gamma_h^j[l_{ij}+1, N](t)) \geq d_h, \text{ or} \\ \max(\underline{z}^i(d(s)), \underline{z}^j(d(t))) - \min(\overline{z}^i(d(s)), \overline{z}^j(d(t))) \geq d_v \end{array}\right. \tag{3.3}$$

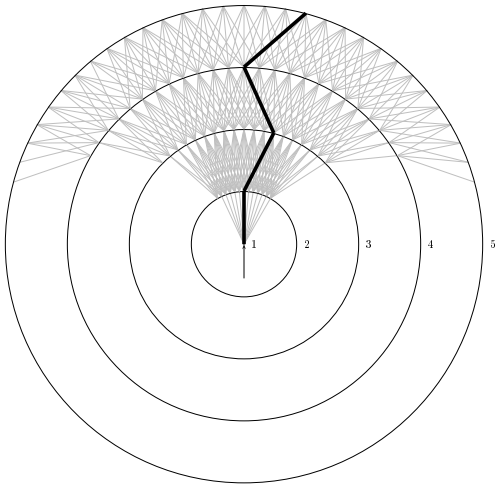


Figure 3.8 – A forbidden path with $\theta_{max} = 30°$.

and

$$\forall i, j \in \{1, ..., N_P\}, j \neq i, \ \widehat{e_m^i e_m^j} \geq \theta_{\min}. \tag{3.4}$$

In this last equation, e_m^i and e_m^j denote the arcs that start on $\mathscr{L}_{ij}$ for routes i and j respectively. Note that the second equation is not always applicable, since routes i and j can belong to two different graphs. Therefore, in this case, l_{ij} is 0, and e_m^i and e_m^j are not defined.

3.3.2.4 Merge points constraint

As stated in the previous section, two merge points cannot be put too close to one another when they belong to the same route. This is because it would induce a requirement for increased attention from the controllers when monitoring the aircraft that use the routes, and an increased need of vectoring on these points (i.e. momentarily changing the aircraft's course to avoid air conflicts). Such a situation would be similar to one where three routes merge together at the same time. This situation is not a problem when designing SIDs, since merging more than two SIDs in our work means dividing the traffic faster. Therefore, there is no need for increased attention from the controllers, since the spacing between aircraft can only increase when a route is divided into several others. However, we chose to keep this constraint active for all routes in our work.

> Two merge points that belong to the same route cannot be closer to each other than d_{m} (3.5)

3.3.2.5 Flight levels constraint

In order to favor the use of the CCO and CDO possibilities, we chose to impose constraints on the level flights. They are stated below:

The number of level flights cannot exceed a given value $n_{\max}^{LF}$
The length of each level flight cannot be less than a given distance $l_{\min}^{LF}$, for this wouldn't make sense in an operational context
The length of each level flight cannot be greater than a given distance $l_{\max}^{LF}$, to allow the aircraft to climb

(3.6)

3.3.3 Objective function

This work aims at optimizing the design of air routes. In the previous paragraphs, we detailed the way in which these routes are represented, and the constraints to take into account for the design. Now, we introduce the optimization function that will allow us to evaluate our design. In our case, the problem is multi-objective in nature. We identified three distinct functions that can be optimized in the process. We detail them in the following paragraphs.

3.3.3.1 The route length

The first and most intuitive criterion is the length of the routes, as shorter routes are often the fastest ones. This may not always be the case, depending on the speed of the aircraft. Indeed, when designing procedures, some velocity constraints can be applied to the routes in order to reduce the margins that are to be taken with the obstacles. Therefore, if a shorter route is velocity-constrained, it may be slower than a longer, unconstrained route. However, we make the assumption here that the speed is not route-dependent, and that all routes are flown at the same speed. Therefore, creating shorter routes also allows airlines to save fuel and to emit less CO2. In this work, a part of the objective is to design a set of routes that have the lowest cumulative length:

$$c_{\text{length}} = \sum_{i=1}^{N_P} F^i \sum_{e \in \gamma_h^i} l(e) \tag{3.7}$$

In this equation, c_{length} is the cost associated to the route length criterion, F^i is the expected traffic flow on route i and $l(e)$ is the length of arc e. This objective function will be referred to as the *route length* criterion in the rest of this document.

3.3.3.2 The graph weight

Another objective that is considered is the *graph weight*, which represents the total space occupied by the graphs. This objective is introduced as a way to make the final solution usable in the current context of air traffic control.

The aim is to orient the method towards merging together the routes that are close to each other. This process allows to be left with as few routes as possible as soon as possible, which makes it easier for the controllers to handle the various air flows. It is equivalent to the problem of finding an optimal Steiner tree, which is an NP-complete problem. This objective that has to be minimized is represented by equation 3.8:

$$c_{\text{weight}} = \sum_{e \in E} \chi(e) l(e) \tag{3.8}$$

In this equation, $\chi(e) = 1$ if e belongs to at least one route, 0 otherwise. It can easily be seen that this part of the objective is different, and sometimes contradictory with the route length criterion. An illustration of this contradiction is provided by fig. 3.9.

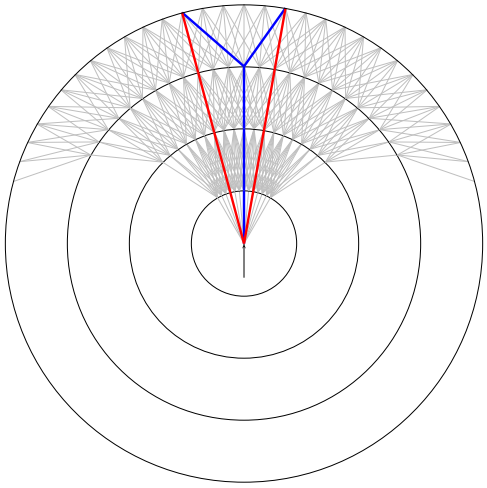


Figure 3.9 – An illustration of the contradiction between best route length (in red) and best graph weight (in blue).

3.3.3.3 The noise abatement

As the air traffic and the population grow, it is more and more difficult to avoid flying over cities. However, this is also more and more required when the routes are designed, as environmental concerns arise. Thus, this criterion has been taken into account as follows in our work: the aircraft must avoid flying over them, but if there is no choice, the impact has to be as small as possible, so the routes must try to fly over the least populated areas.

$$c_{\text{noise}} = \sum_{i=1}^{N_P} F_i \sum_{\tau \in T} \int_0^1 \left(\int_{z^i(\mathrm{d^i(t)})}^{\overline{z}^i(\mathrm{d^i(t)})} c_\tau(\gamma_h^i(t), z) \mathrm{dz} \right) \mathrm{dt} \tag{3.9}$$

where $c_\tau(\gamma_h^i(t), z)$ is the cost of an aircraft flying at altitude z at $\gamma_h^i(t)$ regarding noise emissions. The noise intensity varies with the altitude of the aircraft, and its calculation can take into account many parameters [119]. As

a simplification, in this document, we consider that the nominal noise (noise intensity besides the aircraft) is decreased by 6dB every time the distance to the aircraft is multiplied by 2. The nominal noise at 3 meters is set here at 100dB [120]. The cost $c_\tau(x, y, z)$ for τ being a city is expressed as:

$$c_\tau(x, y, z) = \eta(x, y) \cdot \max\left(\left(100 - 6\frac{\ln \frac{z}{3}}{\ln 2}\right), 0\right) \tag{3.10}$$

A graphical representation of c_τ is provided in fig. 3.10. In fig. 3.10, the function $\eta(x, y)$ is represented on the ground on a 10×10 grid, and varies between 0 and 1. The brightest color corresponds to the highest population density. Above, the cost function is represented, for a flight at a constant altitude of 10,000ft.

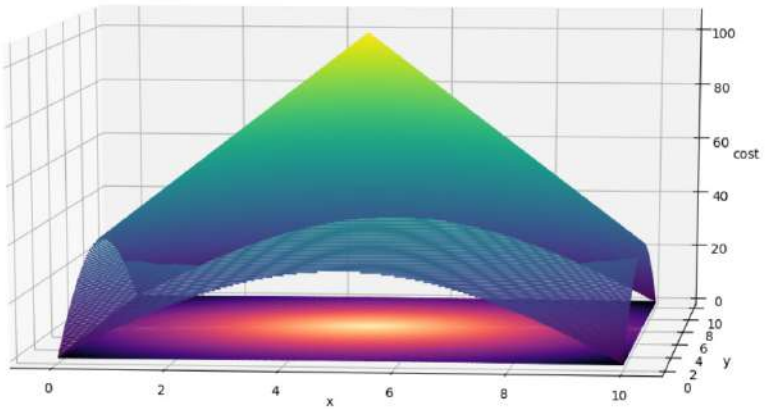


Figure 3.10 – An illustration of the cost function c_τ (equation 3.9) at a constant altitude of 10,000ft. The population density is displayed on the horizontal plan (the brighter the color, the higher the density).

3.3.3.4 The complete optimization problem

As illustrated in the rest of this section, the problem is indeed multi-objective. In order to simplify the optimization process, we chose to combine the three objectives into one, as a linear combination. By taking into account the constraints stated above, the overall optimization problem can be written as follows:

$$\begin{cases} \min & \alpha c_{\text{length}} + \beta c_{\text{weight}} + \gamma c_{\text{noise}} \\ s.t. & \text{Obstacle avoidance constraint (3.1)} \\ & \text{Route separation constraint (3.3)(3.4)} \\ & \text{Limited turn constraint (3.2)} \\ & \text{Merge constraint (3.5)} \\ & \text{Level flights constraint (3.6)} \end{cases}$$

where α, β and γ are chosen by the user and express the relative importance of these criteria.
By choosing a single-objective function, all the criteria are combined into

one. As a result, the solutions may be optimal in neither of these criteria, as, for instance, the route length and the graph weight are contradictory objectives, most of the time: the route length objective tends to make the routes go straight from the center to the exit point while the graph weight criterion will often push the merging points towards the exits. In this formulation, we add together two lengths and a measurement of noise. Therefore, it is not easy to compare the three criteria and to set values for α, β and γ. A solution can be to divide each criterion by an upper bound of this criterion, so as to manipulate percentages, which are comparable. However, finding an upper bound for each criterion, that would preserve the scale of each value is not easy. Another solution would then be to take the noise criterion out of the equation and to integrate it in the optimization problem as a constraint. This solution would allow to deal with a length as the objective function, with two comparable parts (route length and graph weight). However, this solution is not explored in the context of this work.

In this chapter, we established the mathematical modeling for the optimal SID/STAR design problem. The TMA is sampled by the means of concentric layers in the horizontal plan, which provide a graph structure in which a multiple path search can be performed. A route is made up of two parts: a horizontal profile, represented by a path in the graph, and a vertical profile, which provides the range of possible altitudes that an aircraft can attain at each point along the horizontal profile. Numerous constraints, such as obstacle avoidance or the location of the merging points of the routes must be taken into account. The resulting optimization problem is complex, and as a simplification, the three chosen optimization criteria are mixed together into a single objective function. This choice allows to apply mono-objective resolution methods. In the next chapter, we present two resolution methods: a method based on the dynamic programming principle, and the other based on the Simulated Annealing metaheuristic. The strengths and weaknesses of both methods are analyzed, and the details of the resolution process are presented for each one of them.

Chapter 4

Resolution approach

In this chapter, we describe the choices that were made to tackle the problem of SID/STAR optimization. First, we present a deterministic resolution method based on the principle of dynamic programming. Its performances and limitations are emphasized. Then, we introduce a second approach based on the use of a metaheuristic: the Simulated Annealing, and the way in which it was adapted to our problem. We then present in detail each step of the process: the route generation with an adaptation of the Bellman-Ford algorithm, then the design of several routes, and the solution evaluation process. We put an emphasis on the different ways to generate a neighbor solution in the SA for our problem.

4.1 The dynamic programming principle

In order to compute the routes, a first option is to use an exact approach such as the dynamic programming principle, as it is used in many shortest path search problems. This category of algorithm relies on the fact that any part of a shortest path is itself the shortest possible. For instance, if a path is to be computed between two points A and B, and if the shortest path from A to B passes by a point P, then the path from A to P is the shortest. Therefore, the algorithm needs to keep only the best subsolution at each step in order to find the best final solution. In our work, such an approach could be applied in the graph structures (described in chapter 3), by taking advantage of the structure by layers.
In our case, we have to take the altitude into account in the construction of a solution. In some cases, the routes must avoid obstacles, such as mountains or other routes, for instance. Therefore, in order to be able to compute the optimal set of paths, these constraints must be taken into account. A report on the adaptation of the dynamic programming principle to the optimal constrained path search can be found in [121].

4.1.1 Modeling of the optimal SID/STAR design problem as an optimal shortest constrained path

Under certain hypotheses, that will be detailed later, the problem of optimal SID/STAR design can be described as a problem of shortest constrained path as presented in [121]. In [121], the problem of finding the shortest constrained path under constraints can be described by using the following notations:

- $G = (V, A)$ an oriented graph, where V is the set of vertices, of size n, and $A \subset V \times V$ is the set of edges, of size m;
- $\mathscr{R}$ is the set of resources, of size R;
- To each arc $(i, j) \in A$ is associated a cost c_{ij} and a non-negative resource consumption vector $(t_{ij}^r)_{r=1,\dots,R}$;
- The graph G cannot contain an absorbing circuit, i.e. it cannot contain a subset $B = \{(i_0, i_1), (i_1, i_2) \dots (i_n, i_0)\} \subset A$ such that $\sum_{j=0}^{n} c_{i_j i_{(j+1) \bmod n+1}} < 0$;
- A path $P_{x \to y}$ from vertex x to vertex y is a sequence of arcs: $P_{x \to y} = \cup_{i=1}^{p} \{(u_i, v_i)\}$ such that $(u_i, v_i) \in A$ for all i, $u_{i+1} = v_i$, $u_1 = x$ and $v_p = y$;
- The length of the path $P_{x \to y}$ presented above is p;
- The cost of the path $P_{x \to y}$ presented above is defined as $C(P_{x \to y}) = \sum_{(u,v) \in P_{x \to y}} c_{uv}$.

The problem of shortest constrained path then consists in finding the path with the minimum cost between two given vertices, that meet the resource constraints. These constraints can be expressed as follows: to each vertex i in V are associated R *resource windows*, denoted $[a_i^r, b_i^r], r = 1, \dots, R$. Each window $[a_i^r, b_i^r]$ represents the amount of resource r that can be used before reaching vertex i. Here, we examine the case where waiting is not permitted, which means that the amount of resource r consumed upon arrival at vertex i cannot be less than a_i^r. We denote $T_{x \to y}$ the vector of resource consumption: the r^{th} component of $T_{x \to y}$ is associated to a path $P_{x \to y}$ and represents the total amount of resource r consumed by taking this path. The resource consumption constraint can then be written as:

$$\forall j \in P_{x \to y}, \forall r \in 1, \dots, R, a_j^r \leq T^r(P_{x \to j}) \leq b_j^r \tag{4.1}$$

where j is a vertex through which $P_{x \to y}$ passes, and $T^r(P_{x \to j})$ is the consumption of resource r upon arrival at vertex j when taking the path $P_{x \to y}$:

$$\forall r = 1, \dots, R, T^r(P_{x \to j}) = \sum_{(u,v) \in P_{x \to j}} t_{uv}^r \tag{4.2}$$

.
We can apply this formalism, and the results obtained with it in [121], to the optimal SID/STAR design problem, by making the following assumptions for the latter:

- We consider only one resource, which is the altitude. Therefore, with the previous notations, $R = 1$;
- We assume that at each point, the range of authorized altitudes is an interval (which in reality is not always the case: it can be a set of intervals);
- In order to simplify the notations, we consider that $\alpha_{\min} = \alpha_{\max} = \alpha$. Therefore, there is only one possible value for the altitude at a given point on the path.

We construct the graph $G = (V, A)$ described above by applying the following steps:

- We consider the graph $G = (V, E)$ built in section 3.2.1;
- We create a set E' of arcs, which is a duplicate of E;
- With $l(e)$ the length of an arc e, we set the resource consumption (the altitude) of each arc as follows:
  - For all arcs e in E: $t_e^1 = \alpha l(e)$. These arcs are those on which no level flight is applied;
  - For all arcs e' in E': $t_{e'}^1 = 0$. These arcs are those on which a level flight is applied;
- We create the set $A = E \cup E'$;
- For each arc $a \in A$, we set its cost: $c_a = l(a)$;
- The results of [121] can be applied on the graph $G' = (V, A)$.

As mentioned in [121], finding the shortest constrained path is a NP-hard problem, even with a single resource to be taken into account.
A way to tackle this problem is to use the principle of dynamic programming described above, that states that any subsequence of an optimal sequence is itself optimal. In dynamic programming approaches, one method to solve the problem is to create and update *labels*. A label is a vector representing a path, each coordinate of which represents either the cost associated to this path or its consumption of the resources. For instance, in the case studied here, a label is a couple of values (c, z) where c is the cost of the path (its length), and z is the altitude reached at the end of this path. Note that to each possible path between two vertices in $G = (V, A)$ corresponds a label. In particular, to each vertex can be associated a label, corresponding to the path between the *center* of the graph (see section 3.2.1 for the definition of the center) and this vertex, as long as such a path exists.
We define the notion of *dominance* of a label over another in the following way: a label (c, z) *dominates* a label (c', z') ($(c, z) \succ (c', z')$) if and only if:

- $c \leq c'$, and
- $z \geq z'$, and
- one of these inequalities is strict.

This criterion favors paths that allow for a quick climb. It allows to:

- reach the en-route sector at a sufficient altitude;
- avoid conflicts with ground obstacle as much as possible;
- reduce the number of level flights, which is beneficial in the context of Continuous Climb Operations.

However, this criterion can also lead to poor results when considering forbidden zones that must be flown under, since it favors quick climbs. Taking these into account would require a more complex model and dominance criterion. Therefore, we can temporarily discard forbidden zones and their effects on the route design in order to keep this section more simple, as the results will not differ in their interpretations.
The set of all non-dominated labels (i.e. all labels that no other dominates) is called the *Pareto front* of the solution.
In order to find the optimal solution in the SID/STAR design problem, one can apply Algorithm 5 from [121], that presents a solution for acyclic graphs, which corresponds to the case of our study. This adaptation is described by Algorithm 3.

Algorithm 3 The algorithm based on the dynamic programming principle to find the shortest constrained path between the center of the graph of section 3.2.1 and one arrival point. It works by computing all possible labels for each reachable vertex in the graph, and then by keeping only the Pareto front of labels for each.

Require: An acyclic graph $G = (V, A)$ with a resource constraint of the type $[z_{\min}, z_{\max}]$ associated to each vertex in V; a starting altitude z_{start} at the center, an arrival vertex.

We denote $\text{LAB}(l, i)$ the set of all labels associated to the i^{th} vertex of the l^{th} layer (see section 3.2.1 for the definition of layers). One such label is of the form (c, z).
Initialization: Set $\text{LAB}(1, 1) \leftarrow (0, z_{\text{start}})$.
for l from 2 to $N_{\mathscr{L}}$ **do**
    for i from 1 to N the number of vertices per layer **do**
        $\text{LAB}(l, i) \leftarrow \emptyset$
        for each arc $(v_{l-1,j}, v_{l,i})$ in A **do**
            Denote c_{ji} the cost of this arc, z_{ji} its altitude variation.
            for each $\text{Lab} \in \text{LAB}(l-1, j)$ **do**
                if $v_{l,i}.z_{\min} \leq \text{Lab}.z + z_{ji} \leq v_{l,i}.z_{\max}$ **then**
                    $\text{LAB}(l, i) \leftarrow \text{LAB}(l, i) \cup \{(\text{Lab}.c + c_{ji}, \text{Lab}.z + z_{ji})\}$
                end if
            end for
        end for
        $\text{LAB}(l, i) \leftarrow$ the Pareto front of $\text{LAB}(l, i)$
    end for
end for
return the path corresponding to the label of minimum cost at the arrival vertex.

4.1.2 Complexity analysis

We can analyze the complexity of the resolution method provided by Algorithm 3. The algorithm operates on the graph $G' = (V, A)$ constructed above, in section 4.1.1. We denote N the number of vertices per layer, and $N_{\mathscr{L}}$ the total number of layers.

4.1.2.1 Spatial complexity

We begin by determining the space that is required to carry out the procedure. Any vertex in the graph has $\mathcal{O}(N)$ predecessors, except for the first and the second layers (the only vertex on the first layer, which is the center, has 0 predecessor, and any vertex on the second layer has at most 1 predecessor: the center).
Then, we compute the maximum number of labels held by any vertex in the graph. This number is equal to the size of the Pareto front of solutions for the optimal path search problem between the center and this vertex. In the worst case, this Pareto front is the size of the number of possible paths from the center to the vertex (which means that no path is dominated). By

recurrence:

- a vertex on layer $\mathscr{L}_2$ holds two labels (one for the arc from the center to this vertex without a level flight, and one for the arc with the level flight, see the definition of the set A of arcs in section 4.1.1);
- a vertex on layer $\mathscr{L}_3$ holds $\mathcal{O}(N)$ labels;
- a vertex on layer $\mathscr{L}_i$ with $i \geq 2$ holds $\mathcal{O}(N^{i-2})$ labels: two for each label of each vertex on the previous layer.

In order to obtain the total spatial complexity, we sum over all vertices:

$$\begin{aligned}\sum_{L=1}^{N_{\mathscr{L}}} N \times N^{L-2} &= \sum_{L=0}^{N_{\mathscr{L}}-1} N^L \\ &= \frac{N^{N_{\mathscr{L}}} - 1}{N-1} \\ &= \mathcal{O}(N^{N_{\mathscr{L}}-1})\end{aligned}$$

This result is intuitive enough: the space needed in the worst case is about the total number of combinations of vertices, one vertex per layer. Thus, this complexity is exponential in the number of layers.

4.1.2.2 Time complexity

In order to compute the time complexity of the method, we analyze the path finding process given by algorithm 3. It works by propagating to a vertex the labels of all of its predecessors (lines 6-13), and then by finding the Pareto front of the resulting set (line 14). If we consider a vertex $v_{i+1,j}$ on layer $\mathscr{L}_{i+1}, i \geq 2$, then it has $\mathcal{O}(N)$ predecessors, each propagating $\mathcal{O}(N^{i-2})$ labels, based on the analysis of the previous paragraph. Therefore, the time required to propagate all labels to $v_{i+1,j}$ from its $\mathcal{O}(N)$ predecessors is $\mathcal{O}(NN^{i-2}) = \mathcal{O}(N^{i-1})$.
Then, we must compute the Pareto front for this vertex. By the dominance criterion stated in section 4.1.1, we know that each label has to be compared to every other label. There are $\mathcal{O}(N^{i-1})$ labels at vertex $v_{i+1,j}$, so comparing one label to all others is done in $\mathcal{O}(N^{i-1})$ time. This operation has to be carried out for each label. The time complexity to compute the Pareto front for one vertex is then $\mathcal{O}(N^{i-1} \cdot N^{i-1}) = \mathcal{O}(N^{2(i-1)})$.
Finally, in order to find the optimal path between the center and a vertex located on layer $\mathscr{L}_{N_{\mathscr{L}}}$, this operation must be done for each vertex on each layer:

$$\begin{aligned}\sum_{L=1}^{N_{\mathscr{L}}} N \times N^{2(L-1)} &= \sum_{L=1}^{N_{\mathscr{L}}} N^{2L-1} \\ &= \frac{1}{N} \times \frac{N^{2N_{\mathscr{L}}} - 1}{N^2 - 1} \\ &= \mathcal{O}(N^{2N_{\mathscr{L}}-3})\end{aligned}$$

As for the space complexity, the time complexity is exponential with the number of layers. These results, although they are given for the worst case

scenario, should be close to the expected time and space complexity, given two reasons:

- Apart from the first layers, and depending on the way that the layers are sampled and the maximum authorized turn angle, it is very likely that every vertex has an important number of predecessors, of the order of $N/3$ for the test cases presented in this thesis;
- The dominance criterion given in 4.1.1 implies to keep a large number of labels for each vertex, since the cost and altitude of a path are proportional to one another (with small variations depending on the level flights).

Therefore, the expected time and space complexities should be close to their worst case evaluation, which is exponential, and therefore not suitable for large instances, which are the subject of this thesis. In order to decrease these complexities, a solution is to use heuristics.

4.1.3 Dynamic programming based heuristic for designing one route

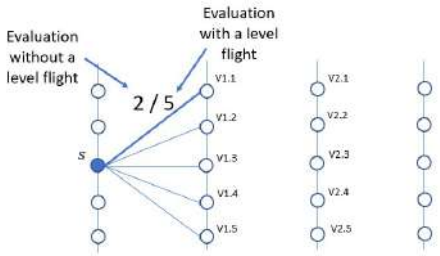
4.1.3.1 Without preprocessing

A method to reduce the space and time complexity of the method can be to consider only the cost of a path for the design, and to consider the minimum and maximum altitudes only when an obstacle is encountered. The corresponding method is described by Algorithm 4 and illustrated in fig. 4.1. This method differs from the one developed in [121] and adapted in the previous section in that only one label is kept for each vertex, representing the cost of each path. By keeping only the cost for each path, we take this method back to a classic path search, only checking that the constraints are met a each label's creation. By taking into account the structure in layers, and since the edges are always oriented from a layer i to the layer $i+1$, this algorithm has a time complexity in $\mathcal{O}(\|A\|)$, which allows to tackle rather large instances.

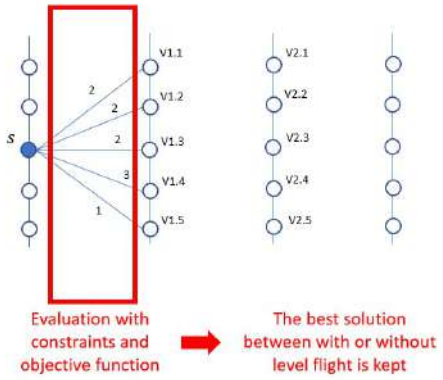
Algorithm 4 The dynamic programming algorithm for the SID/STAR design problem. In this algorithm, the cost of each edge is computed in two configurations: with and without a level flight (lines 11-13). The final cost of the edge is the minimum between the two, and the corresponding configuration is remembered (line 14). Each ending vertex of the edges has then its cost updated whenever its current cost is higher than the cost of a previous vertex, to which is added the cost of the edge (lines 15-22). This process is done for all vertices, layer by layer (lines 9-10).

Require: A starting vertex s, a list of layers L composed of vertices, to which are attached edges, a minimum and a maximum slope of climb/descent, resp. $\alpha_{\min}$ and $\alpha_{\max}$. The set of all vertices is denoted V, the set of all edges is denoted E.

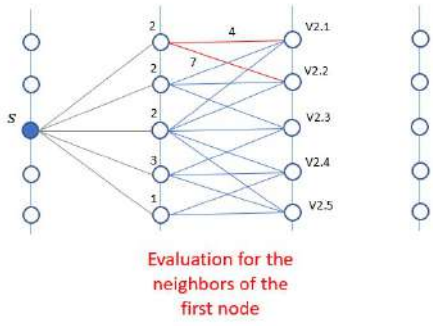
1: Initialization: Create an array d of 'distances' the size of V such that $d[v]$ contains the distance from s to v. Create an array p of predecessors such that $p[v]$ contains the predecessor vertex of v.
2: **for** each vertex v in V **do**
3: $d[v] = \infty$
4: $p[v] = \text{null}$
5: **end for**
6: $d[s] = 0$
7: s.minimumAltitude = 0
8: s.maximumAltitude = 0
9: **for** i from 1 to L.size **do**
10: **for** each vertex u on L_i **do**
11: **for** (u, v) in E **do**
12: compute two surfaces S_1 and S_2. S_1 is the 3D extension of (u, v) obtained applying $\alpha_{\min}$ and $\alpha_{\max}$ on the length of (u, v), starting respectively at the minimum and maximum altitudes of u. S_2 is obtained with the same method, by applying a level flight on (u, v).
13: set c_1 and c_2 the costs of S_1 and S_2 respectively, according to the constraints and objective function.
14: set $c = \min(c_1, c_2)$ and S the corresponding surface.
15: **if** $d[u] + c < d[v]$ **then**
16: $d[v] = d[u] + c$
17: $p[v] = u$
18: **if** S is the version with a level flight **then**
19: set v.levelFlightLength = u.levelFlightLength + distance2D(u,v)
20: **else**
21: set v.levelFlightLength = u.levelFlightLength
22: **end if**
23: **end if**
24: **end for**
25: **end for**
26: **end for**
27: **return** p, d



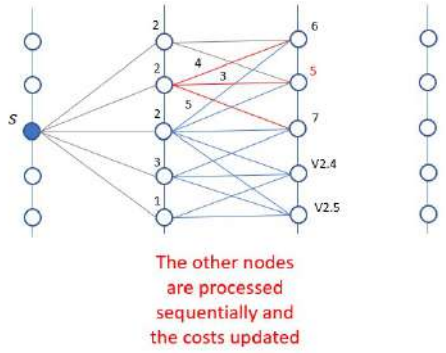
(a) The cost of each neighboring node is computed by taking all constraints and the objective function into account. Two costs are computed: one by considering a level flight on the arc, and the other without.



(b) The cost kept for the neighbor is the minimum between the two computed previously. The costs of all neighbors of the starting node are computed in the same way.



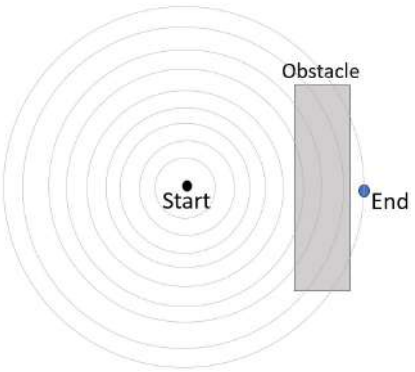
(c) Once all costs for the first neighboring layer have been computed, the same evaluation begins for the second layer.



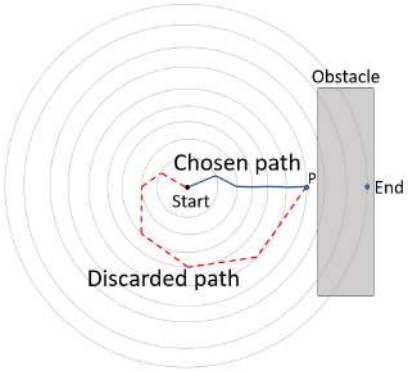
(d) The costs of the neighbors are updated in the same fashion as in the original Bellman-Ford algorithm. The process goes on layer by layer until all costs have been computed in the graph.

Figure 4.1 – The first iteration for the dynamic programming algorithm. All the neighbors of the node are visited, but only a part of them is selected for the next steps.

However, in three dimensions, applying this method without prior preprocessing can lead to the impossibility to find a solution in some particular cases. If we take for instance the situation illustrated in fig. 4.2a, in which an obstacle is to be flown over by making a detour so as to gain enough altitude to pass over it, this method is not able to find a feasible solution, as illustrated in fig. 4.2b. Indeed, by construction, this method will only retain for each node the least expensive way to attain it, based on the information gathered by processing previous nodes. Therefore, it does not take into account the constraints yet to come, nor retains the other possibilities to attain the node. In the example given here, this translates into being unable to find a path that flies over the obstacle when there might be one, because only the shortest path to each node is kept, and the attained altitude is too low to pass the obstacle.



(a) A situation where a detour is necessary to be able to pass over the obstacle.



(b) The default behavior of the method is to build the paths progressively, by keeping only the best current solutions for each node. In this case, the path chosen to attain the point P (solid line) is not long enough to allow for passing over the obstacle. The path that would allow it (in dashed line) was discarded, as it was not the best possible subpath to P.

Figure 4.2 – A situation where the naive heuristic leads to unfeasible solutions.

4.1.4 With preprocessing: imposing boundary values for the minimum and maximum altitude

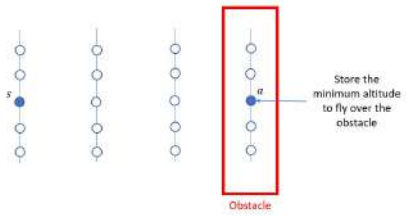
The issue raised in the previous section, regarding the possibility to end up in a dead end due to the lack of visibility on the constraints, can be partly resolved by applying the principle of dynamic programming, in a reverse way (in our case: from the entry/exit point of the TMA towards the center of the graph), so as to propagate the altitude constraints on all nodes on the graph before applying the path search algorithm. This process is described

by Algorithm 5, and illustrated by fig. 4.3. This preprocessing phase allows a later path search algorithm to find paths otherwise discarded by the first heuristic. However, it also creates a dependency on the entry/exit point processed. Therefore, each route must be computed individually, which doubles the computing time of a solution in the case where one route is to be found, and multiplies the computing time of a solution by roughly two times the number of routes to compute in a general case (instead of running the algorithm only once for the whole solution, it is run twice for each route). This increase in computation time can be critical in instances where numerous edges must be processed. In this preprocessing step, some vertices may be discarded for the subsequent path search (lines 25-29 in Algorithm 5). This situation happens when the constraints on the minimum and maximum altitude lead to an unfeasible situation, where the minimum required altitude is higher than the maximum authorized altitude. This idea originates from Algorithm 1 in [121] and allows to overlook some of the infeasible solutions during the path search.

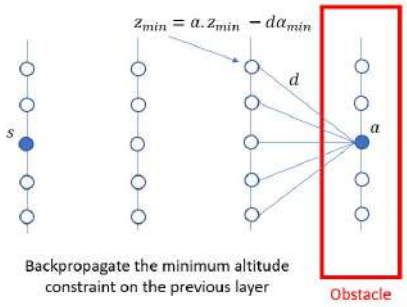
Algorithm 5 The preprocessing method to take altitude constraints into account. This algorithm provides each node of the graph with additional constraints. It begins by computing the requirements of altitude for the arrival point (lines 5-9). Then, these altitude constraints are propagated towards the center (lines 11-12). This is done by updating the minimum possible altitude at the vertex (lines 13-19), and the maximum possible altitude at the vertex (lines 20-22).

Require: The starting vertex s of a route, its arrival vertex a, a list of layers L composed of vertices, to which are attached edges, a minimum slope of climb/descent $\alpha_{\min}$, the altitude constraints imposed by the layout on each vertex, respectively denoted $z_{\min}^{\text{obstacles}}$ and $z_{\max}^{\text{obstacles}}$. The set of all vertices is denoted V, the set of all edges is denoted E.

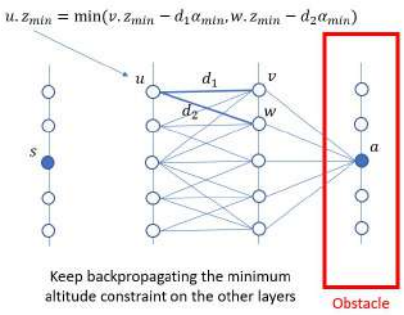
1: Initialization: Create two arrays $z_{\min}$ and $z_{\max}$ of minimum and maximum altitudes both the size of V such that $z_{\min}[v]$ and $z_{\max}[v]$ contain resp. the minimum and maximum authorized altitudes upon arrival at v starting from s to attain a.
2: **for** each vertex v in V **do**
3: $z_{\min}[v] = +\infty$, $z_{\max}[v] = -\infty$
4: **end for**
5: **if** $a.z_{\min}^{\text{obstacles}}$ or $a.z_{\max}^{\text{obstacles}}$ is not a null value **then**
6: $z_{\min}[a] \leftarrow a.z_{\min}^{\text{obstacles}}$ and/or $z_{\max}[a] \leftarrow a.z_{\max}^{\text{obstacles}}$
7: **else**
8: $z_{\min}[a] \leftarrow +\infty$, $z_{\max}[a] \leftarrow -\infty$
9: **end if**
10: Set a as processed
11: **for** i from L.size to 2 **do**
12: **for** each edge (u,v) in E such that v is on L_i and v is processed **do**
13: **if** $d_{\min}[v]$ is finite **then**
14: Set d the length of the proj. of (u,v) on the horizontal plane
15: $z_{\min}[u] \leftarrow \min(z_{\min}[u], z_{\min}[v] - d\alpha_{\min})$
16: **end if**
17: **if** $u.z_{\min}^{\text{obstacles}}$ is not a null value **then**
18: $z_{\min}[u] \leftarrow \max\left(z_{\min}[u], u.z_{\min}^{\text{obstacles}}\right)$ if $z_{\min}[u]$ is finite, $u.z_{\min}^{\text{obstacles}}$ otherwise
19: **end if**
20: $z_{\max}[u] \leftarrow \max(z_{\max}[u], z_{\max}[v])$
21: **if** $u.z_{\max}^{\text{obstacles}}$ is not a null value **then**
22: $z_{\max}[u] \leftarrow \min\left(u.z_{\max}^{\text{obstacles}}, z_{\max}[u]\right)$ if $z_{\max}$ is finite, $u.z_{\max}^{\text{obstacles}}$ otherwise
23: **end if**
24: **end for**
25: **for** all vertices u on layer i **do**
26: **if** $z_{\min}[u]$ and $z_{\max}[u]$ are finite and $z_{\min}[u] > z_{\max}[u]$ **then**
27: Discard vertex u and all the edges connected to it
28: **end if**
29: **end for**
30: **end for**
31: **return** $z_{\min}$, $z_{\max}$



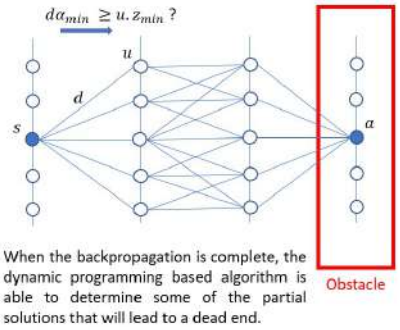
(a) The minimum altitude to pass over the obstacle is deduced from the height of the obstacle beneath a.



(b) All vertices connected to the arrival vertex are updated so that each of them holds the minimum altitude required to be able to pass over the obstacle when it is needed.



(c) The backpropagation of the constraint is carried out until the route start.



(d) When the backpropagation is complete, the path search algorithm can be run, by taking into account the minimum altitude constraint. The process is exactly the same for the maximum altitude constraint (which occurs when some obstacles have a nonzero lower altitude, such as forbidden zones).

Figure 4.3 – The backpropagation method prior to the dynamic programming based path search algorithm on the graph allows it to avoid some dead ends when computing the routes.

However, on top of the increase in computation time, applying Algorithm 5 is not enough to guarantee the optimality of the solution. This is due to the fact that this method stores only the most favorable constraint for the subsequent path search. Therefore, it improves its capacity to find a solution, but finding the optimal solution is subject to the same problem as stated before, in which a first part of a solution is computed, which is optimal for the given criterion and for the subpath under consideration, but will lead to a suboptimal solution. This case is illustrated by fig. 4.4. In this example, two subpaths from the starting point s and a point P are illustrated, with the arrival point a behind an obstacle. The obstacle has two different heights: low in blue and moderate in red. During the preprocessing phase, as it is described in Algorithm 5, a minimum altitude is associated to the point P. This minimum altitude corresponds to the low obstacle. We assume that the altitude reached by the path in solid line is that same minimum altitude. Then, the best admissible path from P to a is over the low obstacle. This path will be chosen by the method, even though the path in dashed line is a better solution to the problem. More importantly, this method still cannot

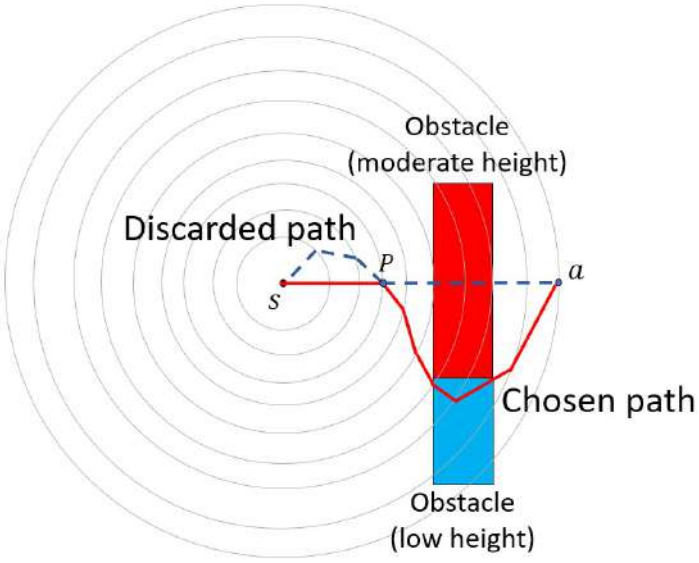


Figure 4.4 – A case where requiring a minimum altitude is not sufficient to find the optimal solution: the path in solid line is chosen over the one in dashed line, even though the latter is shorter.

guarantee to find any solution when there might be one. An example is provided by fig. 4.5. In this simplified example, there are two possible routes: one that passes over a mountain, and one that passes beneath a forbidden zone. With the method described above, and since only the most permissive constraints are backpropagated to each point, point P is not constrained on the minimum altitude due to the backpropagation from the forbidden zone, and P is not constrained in maximum altitude either, due to the backpropagation from the mountain. If we consider that the shortest path from the center to P is too long to allow to pass under the forbidden zone, but too short to allow to pass over the mountain, the algorithm cannot find any solution, even though there might be one (that passes over the

mountain), despite the preprocessing phase.

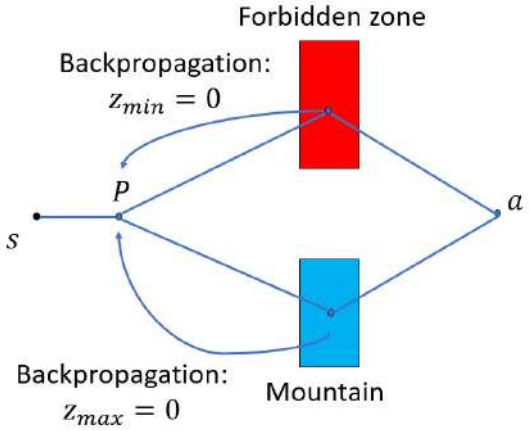


Figure 4.5 – A case where the preprocessing phase is not enough to allow the algorithm to find a solution: the most permissive constraints only are backpropagated from the forbidden zone and the mountain, and the resulting situation leads to the same problem as the case where no preprocessing is applied.

4.1.5 Heuristic for designing several routes

In the previous sections, we presented a method to design one route under altitude constraints. In the next paragraphs, we extend this method to the design of several routes.
A first idea can be to use repeatedly the method for one route until all routes are designed. This idea is illustrated by Algorithm 6. In this case, only one

Algorithm 6 Designing a set of routes sequentially by using the dynamic programming heuristic with preprocessing. All routes start at the center.

Require: The center of the graph c, all entry/exit points of the TMA as vertices $P^1, \ldots, P^{N_P}$, a list of layers L composed of vertices, to which are attached edges, a minimum slope of climb/descent $\alpha_{\min}$, the altitude constraints imposed by the layout on each vertex, respectively denoted $z_{\min}^{\text{obstacles}}$ and $z_{\max}^{\text{obstacles}}$. The set of all vertices is denoted V, the set of all edges is denoted E.

1: Initialization: Create an array r of routes.
2: **for** each entry/exit point $P^i, i = 1, \ldots, N_P$ **do**
3: Apply the preprocessing technique of Algorithm 5 by considering routes $1, \ldots, i-1$ as obstacles
4: Let r^i be the result of Algorithm 4 applied between the center and P^i
5: $r[i] \leftarrow r^i$
6: **end for**
7: **return** r

graph of the form $G = (V, A)$ as described in section 4.1.1 is considered, which means that only one runway (and one orientation of this runway) is taken into account. The method, however, is the same when taking several graphs into account. The most important drawback of this method is that it doesn't take into account the need to separate the merging points of the routes. Indeed, Algorithm 6 shows that all routes depart from the origin. As explained in the previous chapters, such a design is not acceptable in real-life conditions. Therefore, some of the routes must be allocated a new starting point. In order to achieve this, one can proceed as explained in Algorithm 7. The routes are ordered by decreasing order of traffic flow. This will allow to favor a more direct design for the busiest routes. Then, for each route R except the first one, we search for the point P with a maximum curvilinear abscissa such that P is too close to another route. We denote L the layer containing P, and R' the route that P is too close to. The new starting point for R is then the intersection of R' with L. The process is illustrated by fig. 4.6.

(a) The initial solution for the concurrent merging problem. Routes R and R' are too close, from the center to layer L.

(b) The new start for route R is the intersection of R' with L.

Figure 4.6 – The heuristic for solving the concurrent merging problem. The routes start on the route closest to them, on the last layer on which the routes were too close to each other.

4.1.6 Motivations for a metaheuristic based approach

As demonstrated in the last paragraphs, the heuristic constructed from the dynamic programming approach has several severe drawbacks in the scope of this work. Firstly, the optimality of the solution is not automatically verified. This problem could be overcome by keeping a Pareto front of labels, but in our case, it would take too much time to find a solution, as demonstrated in section 4.1.2.2. Also, as for the routes in one graph, when several graphs are to be taken into account, these must be processed sequentially. Therefore, in order to find the optimal solution, one should also test all combinations for the order of generation of the graphs, and routes.

Even so, the optimality cannot be guaranteed. For instance, optimality cannot be achieved in a case where two graphs are to be considered and the optimal solution involves a level flight in both, in order to avoid a route from the other (see an illustration of this case in fig. 4.7). Such a solution can only be found if both graphs are processed at the same time, or by introducing random behaviors. Finally, and most importantly, the method based on the

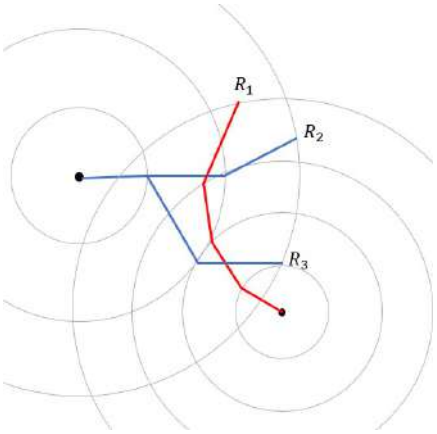


Figure 4.7 – A case where two graphs must be processed at the same time in order to find the optimal solution. Route R_1 must have a level flight in order to pass below R_3 and route R_2 must have a level flight to pass below R_1.

heuristic is not guaranteed to find a solution, even when one exists (even when taking the preprocessing steps), as explained in section 4.1.4. Despite these drawbacks, this method presents various advantages, listed below:

- It is an efficient way of designing a single route when only one obstacle is to be taken into account.
- It doesn't need any parameter setting, which makes it very easy to use. The user only has to provide the starting and ending point of the route.

These features make the method fairly efficient in cases where only one, or very few routes are to be computed, with few to no obstacles to take into account. However, with the risk of not being able to find any solution when there might exist one, by introducing a heuristic for the choice of the new starting points for the routes, and with the need to process each separate graph in a sequence, this approach is not viable on its own for general cases. This algorithm could be considered to make for a subroutine of a metaheuristc-based approach. In the next sections, we present the approach chosen to solve the optimal SID/STAR design problem, that is based on a meta-heuristic.

Algorithm 7 The heuristic for the design of several routes with the dynamic programming based approach. Lines 2 to 5 build a first solution, with all routes starting on the center. Lines 6 to 15 determine for each route the last route among the previous ones that is too close to the route under consideration. Based on this information, lines 16 to 24 recompute the routes with a new starting point so that the merging constraint as well as the route separation constraint are met.

Require: The center c of the graph, the entry/exit points of the TMA $P^1, \dots, P^{N_P}$, a list of layers L composed of vertices, to which are attached edges, a minimum slope of climb/descent $\alpha_{\min}$, the altitude constraints imposed by the layout on each vertex, respectively denoted $z_{\min}^{\text{obstacles}}$ and $z_{\max}^{\text{obstacles}}$. The set of all vertices is denoted V, the set of all edges is denoted E.

1: Initialization: Create an empty set R of routes, a double-entry array A of size $N_P \times 2$. This array will contain for each index i both the last route that is too close to $\mathscr{R}^i$ and the last layer for which the two routes were too close to each other. All values of A are initialized to -1.
2: **for** i from 1 to N_P **do**
3: Apply the preprocessing algorithm (Algorithm 5) between c and P^i
4: Compute $\mathscr{R}^i$ between c and P^i with Algorithm 4
5: **end for**
6: **for** i from 2 to N_P **do**
7: **for** j from 1 to $N_{\mathscr{L}}$ **do**
8: **for** k from 1 to $i-1$ **do**
9: **if** $\mathscr{R}^i$ and $\mathscr{R}^k$ violate the route separation constraint on layer j **then**
10: $A[i][0] \leftarrow k$
11: $A[i][1] \leftarrow j+1$
12: **end if**
13: **end for**
14: **end for**
15: **end for**
16: $R \leftarrow \mathscr{R}^1$
17: **for** i from 2 to N_P **do**
18: **if** $A[i][1] = -1$ **or** $A[i][1]$ is greater than the number of the layer that contains P^i **or** $A[i][1]$ is greater than the number of the layer that contains $P^{A[i][0]}$ **then**
19: Set the new starting point $s = c$
20: **else**
21: Set the new starting point $s = \gamma_h^{A[i][0]}(\tau_{A[i][1]}^{A[i][0]})$, which is the intersection of $\mathscr{R}^{A[i][0]}$ with the layer $\mathscr{L}_{A[i][1]}$. If s is already a merging point, increase the layer of connection until the intersection of $\mathscr{R}^{A[i][0]}$ with the layer is not already a merging point. If no such point exists, set $s = c$.
22: **end if**
23: Apply the preprocessing algorithm (Algorithm 5) between s and P^i
24: Compute $\mathscr{R}^i$ between s and P^i with Algorithm 4
25: $R \leftarrow \mathscr{R}^i$
26: **end for**
27: **return** R

4.2 The Simulated Annealing algorithm for the SID/STAR design problem

We start by introducing the general method on which our algorithm relies to handle the optimization of SIDs and STARs: the *Simulated Annealing* meta-heuristic. It was first introduced in the early 1980's [122] and is designed to address an optimization problem with a mono-objective function. This method aims at mimicking a process originally found in metallurgy, the *annealing*. It consists in heating a material to a high temperature and then letting it cool down at a slow rate in order, for the molecules, to rearrange in a structure of minimal internal energy. This process is opposed to the *quenching*, which consists in immediately submitting the hot material to a low temperature. It also allows the material to harden, but with a non-optimal arrangement of its molecules (see fig. 4.8). The corresponding

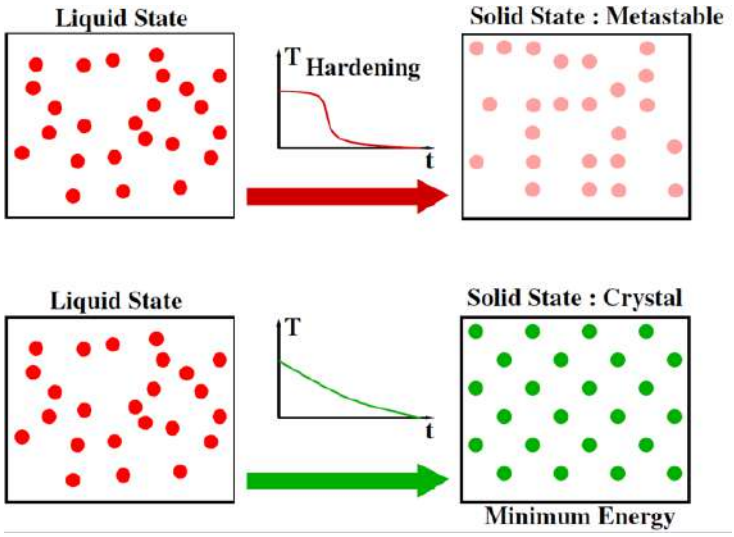


Figure 4.8 – The purpose of the annealing process. Top: quenching, bottom: annealing (taken from [123]).

algorithm is described by Algorithm 8 and works in the following way:

- *Initialization*: A first solution to the problem is computed, that will serve as the initialization of the algorithm. In the mean time, a starting "hot" temperature is chosen. This temperature is set by following these steps:

  1. a fixed initial temperature is set;
  2. a solution to the problem is computed;
  3. a neighboring solution is computed;
  4. the neighboring solution is accepted with a probability depending on the temperature (see the steps in the cooling loop for the probability);
  5. repeat steps 2 to 4 for a fixed number of times;

6. if the number of accepted neighbors is lower than a fixed value, increase temperature and repeat steps 2 to 6.

More detail on the choice of the initial temperature can be found in [123], and the reader can refer to chapter 5 for the specific fixed values used in this work.

- *Cooling loop*:
  - The result of the evaluation of the objective function for the current solution x_i is denoted y_i, and the current temperature of the algorithm is denoted by T.
  - A neighbor x_j of the current solution is randomly generated. This step implies that it is possible to establish a notion of distance between two solutions in the algorithm.
  - The objective function is evaluated for the neighbor solution x_j. The result is denoted by y_j.
  - If y_j is better than y_i, then the neighbor is accepted as the new current solution, and will serve as a starting point for the next iteration of the loop.
  - If y_j is not better than y_i, the neighbor is nonetheless accepted as the new current solution, with a probability of $e^{\frac{y_i - y_j}{T}}$ (4.3). Thus, the probability of accepting a solution worse than the current one decreases with the temperature.
  - The temperature T is decreased.
- *Stopping criterion*: The algorithm stops when it has performed a pre-defined number of iterations, or when the temperature reaches a fixed value. It can also stop earlier in the cases where the optimal value of the objective function is known and the algorithm finds a solution for which the objective function is evaluated to this value.

In our work, the general idea is to gather the data required to build an instance of a problem, and then to apply a SA-based method to compute a solution. The choice of using a meta-heuristic was made because of the large number and the complexity of the constraints to be taken into account, such as the route separation, which make the use of exact approaches more difficult to implement in terms of computation time. In the process used here, the routes are computed one by one in their decreasing order of traffic flow, then the resulting set is evaluated along the optimization function. This process is done iteratively until a stopping criterion is reached. The algorithm is designed to explore the solution space at the beginning, and then focus more and more on the best solutions found towards the end. This exploration behavior, induced by the introduction of randomness in the path search, which will be detailed later in this chapter, allows to generate the routes sequentially, in any given order, without the drawback that this induces in deterministic approaches. In our case, the evaluation process is carried out by putting the generated routes into a simulation containing the

Algorithm 8 The principle of functioning of the SA.

```
1: Build instance
2: Set a starting temperature for the SA
3: Compute an initial solution
4: while stopping criterion isn't reached do
5:     for  a fixed number of iterations do
6:         Compute a neighboring solution
7:         Evaluate the new solution
8:         Select one of the two solutions as reference for the next iteration
9:     end for
10:    Decrease temperature
11: end while
12: return best solution encountered
```

parameters relative to the terrain and the cities. This simulation computes the cost of the given solution in order to estimate its performance in the scope of the problem (see fig. 4.9).

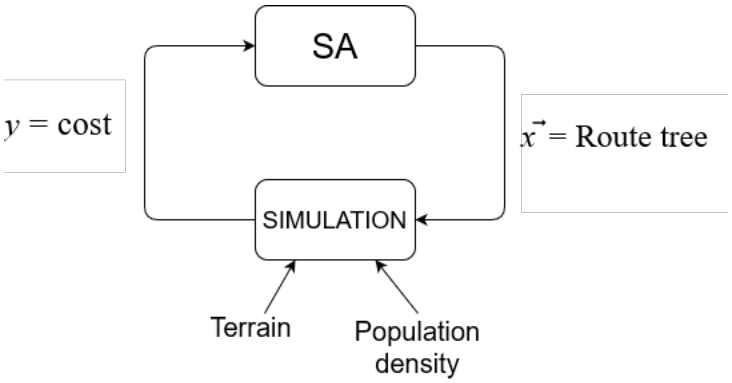


Figure 4.9 – The simplified process for the SA adapted to our case. The solution computed by an iteration of the SA is evaluated by the means of a simulation and the result is used for the subsequent iterations of the SA.

4.3 Generating one route on the graph: the modified Bellman algorithm

In this section, we describe in details the way in which the Simulated Annealing and Bellman-Ford algorithms have been adapted to our problem. The Bellman-Ford algorithm is used to solve the single source shortest path problem in a directed graph [56]. In its original version, it works by updating repeatedly the minimum cost to each vertex in the graph with estimations that are more and more precise at each iteration. The original algorithm is described in Algorithm 9 and illustrated in fig. 4.10. This algorithm iteratively updates the cost of each vertex in the graph by comparing its current cost with the cost of all of its neighbors added to the cost of traveling from the neighbor to the vertex (lines 8-13). This process is done as many

Algorithm 9 The original Bellman-Ford algorithm for the one-to-all shortest path problem in a graph.

Require: A starting vertex s, a list of vertices V, a list of edges E

```
1:  Initialization: Create an array d of distances the size of V such that d[v]
    contains the distance from s to v. Create an array p of predecessors such
    that p[v] contains the predecessor vertex of v
2:  for each vertex v in V do
3:      d[v] = ∞
4:      p[v] = null
5:  end for
6:  d[s] = 0
7:  for i from 1 to V.size-1 do
8:      for (u, v) with weight w in E do
9:          if d[u] + w < d[v] then
10:             d[v] = d[u] + w
11:             p[v] = u
12:         end if
13:     end for
14: end for
15: for (u, v) with weight w in E do
16:     if d[u] + w < d[v] then
17:         Error: there is a negative cycle
18:     end if
19: end for
20: return p, d
```

times as there are vertices (minus one) so that all possible paths within the graph are checked (line 7). This algorithm has a time complexity in $\mathcal{O}(\|V\| \, \|E\|)$.

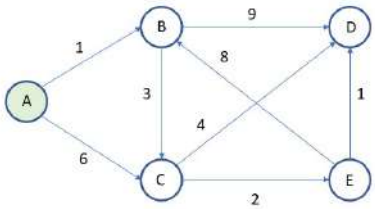
4.3.1 The adaptation of the Bellman-Ford algorithm to our problem

In our version, we modify the Bellman-Ford algorithm to take advantage of the structure in layers. This adapted version is presented in Algorithm 10, which is illustrated by fig. 4.11. In fig. 4.11a, we consider the starting node for a path, as well as its neighbors: V1.1 to V1.5, with their associated costs. At this point, the only *marked* node is s. The table below holds for each vertex its minimal cost (distance) from s, d(v), the preceding vertex in the shortest path from s to this vertex, p(v), and a value true or false that indicates whether the arrival point a is reachable from the vertex under consideration. In the second step (fig. 4.11b), the aim is to select a subset of arcs with minimum cost among the arcs that allow to reach the arrival point a. This subset is denoted E_u, u being the current expanded vertex (so, in fig. 4.11b, $u = s$). In step 3 (fig. 4.11c), the ending vertices of the selected edges are *marked*. The algorithm will then repeat the three steps with each *marked* vertex in the second layer, and proceed through all the layers in the

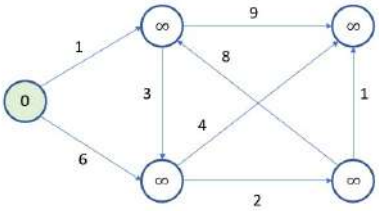
same way until the layer of a is considered. The shortest path from s to a is given by the sequence of predecessors starting at a, taken backwards.
This version of the Bellman algorithm allows for a better computation time, since we don't consider all edges at each iteration but only the edges associated to the vertices of a single layer. When a vertex is not *marked*, none of its edges is considered at all. The other noticeable feature of this modified version is the use of the list E_u containing only a portion of all edges coming out of a given vertex u. This choice was made in order to save computation time by only considering the arcs with the lowest cost. In theory, it puts the optimality of the solution in jeopardy by integrating a greedy behavior. However, when we consider the context of the work, two arguments can be raised in favor of this approach:

- The weights of the edges are their length, as they represent the distance between two points. Therefore, there should be no significant difference between the weight of two edges starting on the same vertex, and so over the whole graph: the weights are uniform.

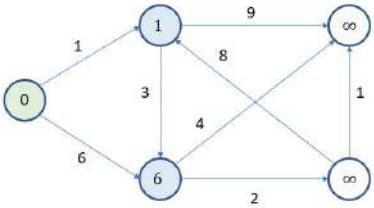
- This modified version of the Bellman-Ford algorithm is used to generate routes between two given points in the search space. In order to allow for exploration of the search space, at each path search, the weight of the edges is changed, in a way that will be explained in the next paragraph. Therefore, we do not, in fact, look for the shortest path, but rather a short, reproducible path. Since the modified Bellman-Ford algorithm is itself embedded in a Simulated Annealing, the latter is the one responsible for choosing the best weights for the edges. Since the algorithm is designed to explore the search space, it is very likely that previously discarded edges will be taken into account in a later iteration.



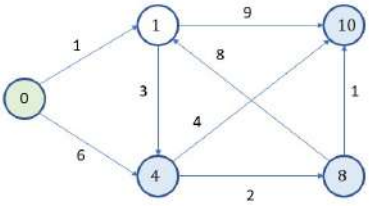
(a) The graph used for the example of the original Bellman-Ford algorithm. The starting node is colored in green.



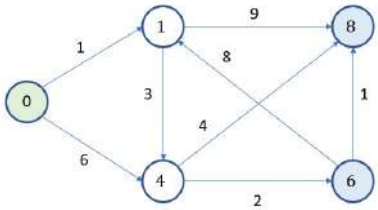
(b) The initial state of the algorithm. The source node is A with cost 0, the rest have an initial infinite cost.



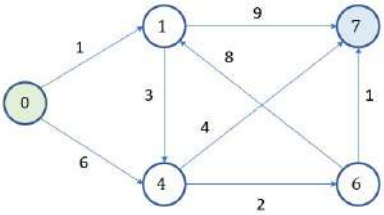
(c) Step 1: A first relaxation of all arcs is performed. The costs of all nodes are updated. There will be as many relaxations as the number of vertices except A. The nodes whose cost is updated are colored in blue.



(d) Step 2: A second relaxation of all arcs is performed. The costs of all nodes are updated. Node B has its cost unchanged.



(e) Step 3: A third relaxation is performed.



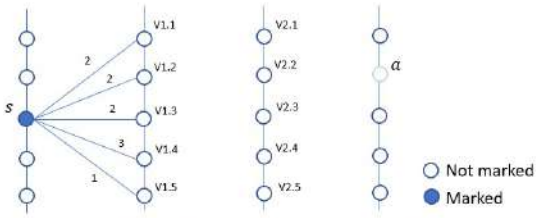
(f) Step 4: A fourth relaxation is performed. Only node D has its cost changed.

Figure 4.10 – An example of run of the Bellman-Ford algorithm. The letters are replaced by the cost for a better readability. The starting node is colored in green ; the nodes whose cost is updated are colored in blue at each iteration.

Algorithm 10 The Bellman-Ford algorithm in the SA method. The original algorithm is modified to take the layer structure into account. The process is sped up by taking into account only the edges that allow to reach the destination (line 11), by expanding only the nodes previously visited (lines 8,10,17) and by visiting only a promising set of neighbors for each node (line 11).

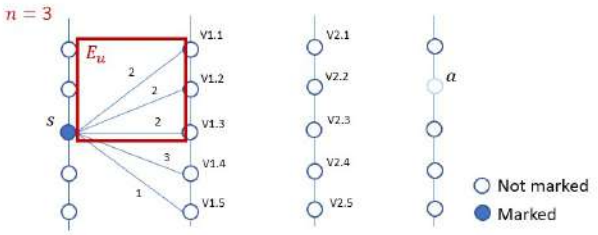
Require: A starting vertex s, an arrival vertex a, a list of layers L composed of vertices, to which are attached edges. s is located on the first layer and a on the last layer, an integer n giving the maximum number of edges to take into account for each vertex expanded

1: Initialization: Create an array d of distances the size of V the set of all vertices such that $d[v]$ contains the distance from s to v. Create an array p of predecessors such that $p[v]$ contains the predecessor vertex of v.
2: **for** each vertex v in V **do**
3: $d[v] = \infty$
4: $p[v] = \text{null}$
5: Tag v as *not marked*
6: **end for**
7: $d[s] = 0$
8: Tag s as *marked*
9: **for** i from 1 to L.size-1 **do**
10: **for** each vertex u on $L[i]$ tagged as *marked* **do**
11: create a list E_u containing the n edges (u, v) of E with the lower weight and such that a is tagged as *true* for v. If there are fewer than n such edges, E_u contains all of them
12: **for** each (u, v) with weight w in E_u **do**
13: **if** $d[u] + w < d[v]$ **then**
14: $d[v] = d[u] + w$
15: $p[v] = u$
16: **end if**
17: Tag v as *marked*
18: **end for**
19: **end for**
20: **end for**
21: **return** p, d



Vertex	s	V1.1	V1.2	V1.3	V1.4	V1.5
d(v)	0	infinity	infinity	infinity	infinity	infinity
p(v)	null	null	null	null	null	null
tag a	true	true	true	true	true	false

(a) The initial state of the algorithm. Only s is *marked*.



Vertex	s	V1.1	V1.2	V1.3	V1.4	V1.5
d(v)	0	infinity	infinity	infinity	infinity	infinity
p(v)	null	null	null	null	null	null
tag a	true	true	true	true	true	false

(b) Expansion of the promising arcs. V1.5 is not expanded because a is unattainable.

Vertex	s	V1.1	V1.2	V1.3	V1.4	V1.5
d(v)	0	2	2	2	infinity	infinity
p(v)	null	*s*	*s*	*s*	null	null
tag a	true	true	true	true	true	false

(c) The state of the algorithm before the second iteration: the ending vertices of the selected arcs are *marked*. Only marked nodes on layer 2 will be expanded.

Figure 4.11 – The first iteration for the modified Bellman-Ford algorithm. All the neighbors of the node are visited, but only a part of them is selected for the next steps.

4.3.2 The management of the edges' weight

Each individual path is computed with an adapted version of the Bellman-Ford algorithm, with a carefully chosen cost for the arcs, as the algorithm has to be able to explore various possible paths for two given starting and ending points. This last requirement implies that the cost of the edges can vary during the execution of the algorithm. Otherwise, for two given points, the result of the search would always be the same. To achieve this, the algorithm biases the costs of the edges in the graph for each path to be computed.This process is done by the SA. A number (the bias) between -1 and +1 is associated to each merging layer. A negative bias will increase the costs of the arcs 'on the right' of each node until the next merge layer, while a positive bias will increase the costs 'on the left' (see Figure 4.12). The closer the bias is to -1 or +1, the sharper the turn

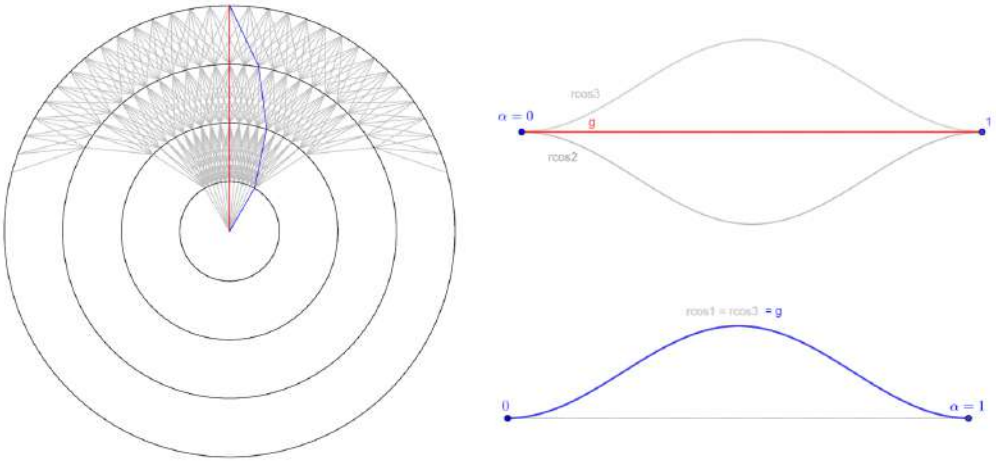


Figure 4.12 – Two paths to the same exit and the associated biais functions. The shape of the paths is similar to the associated shape of the bias function g on $[0\ ,\ 1]$.

will be, while a null bias will favor the straightforward paths. By resetting the costs of the arcs at each path search, it is possible to have a complete exploration of the graph while keeping a way to recreate any path, given its start and end points along with the biases. The way in which the biases are applied to the edges of the graphs is described in Algorithm 11, and illustrated in fig. 4.13. In a nutshell, this piece of algorithm does, for each node and given biases, the following:

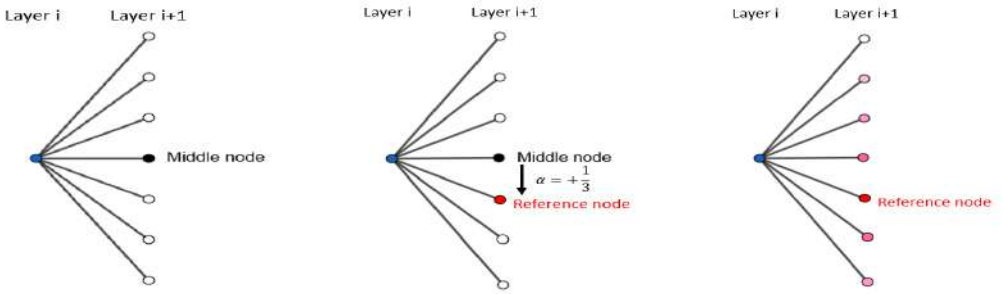
Algorithm 11 The arcs valuation algorithm.

Require: A starting vertex s, an exit vertex e, layers $\mathscr{L}_1 ... \mathscr{L}_n$ where s belongs to $\mathscr{L}_1$ and e belongs to $\mathscr{L}_n$, an array A of size $n-1$ containing float values between -1 and $+1$.

Initialization: Set all arcs costs to their length. Tag s as *marked*. Choose an expansion coefficient α.
for each $\mathscr{L}^i$ **do**
  for each *marked* vertex v on $\mathscr{L}_i$ **do**
    Let n_v be the number of arcs starting on v
    Number all arcs (v, w) from 1 to n_v from left to right and *mark* each w
    Let m be $\lfloor (1 + A[i]) \frac{n_v - 1}{2} \rfloor + 1$
    Arc number m has its cost unchanged
    Let $G = Max(m, n_P - m)$
    for all arcs e_{k_1} on the right of arc m **do**
      Multiply the cost of e_{k_1} by $1 + \frac{k_1 - m}{G}(\alpha - 1)$
    end for
    for all arcs e_{k_2} on the left of arc m **do**
      Multiply the cost of e_{k_2} by $1 + \frac{m - k_2}{G}(\alpha - 1)$
    end for
  end for
end for

- Find the middle node among its successors (Fig. 4.13a);
- Find the node corresponding to the assigned coefficient. This coefficient is the same for all nodes between two given merging layers. The gap between the middle node and the node to find is proportional to the assigned coefficient (which is in $[-1, +1]$). The sign of the coefficient indicates the direction of the gap relatively to the middle node (right or left). For instance (see fig. 4.13b), if there are 3 nodes on the right of the middle node, and the coefficient is $\frac{1}{3}$, then the node to find is the first one on the right of the middle node. This new node is called the *reference node*;
- The reference node has its cost unchanged;
- The algorithm iteratively increases the costs of the nodes neighboring the reference node. The farther the node from the reference node, the more its cost is increased (fig. 4.13c).

It is during this phase that the list E_u in Algorithm 10 is created. Once all edges starting on a given vertex have their new weight set, the n of them with the lowest weight can be put into the list E_u. The coefficients for the cost bias are chosen in a way that helps reducing the zigzag phenomenon. The aim is to generate a sequence of numbers that are coherent with their predecessors and successors. In order to achieve this, the generation is based on the raised cosine function:



(a) Identification of the middle neighbor of a node.

(b) Identification of the reference node.

(c) Valuation of the arcs. The cost to the red node is unaffected. The cost to the white one is heavily affected.

Figure 4.13 – The arcs valuation function. The cost of the arcs is increased proportionally to the distance of their ending vertex to the reference node.

$$\mathrm{rcos}(x, \mu, s) = \frac{1}{2s}\left[1 + \cos\left(\frac{x-\mu}{s}\,\pi\right)\right] \text{ on } [\mu - s, \mu + s] \tag{4.4}$$

The method works as follows, for a given starting vertex o (origin) and a given ending vertex a (arrival):

- A number $\alpha \in [0, 1]$ and two signs $\mathrm{sgn}_1, \mathrm{sgn}_2 \in \{-1, +1\}$ are randomly chosen;
- Three amplitude values A_1, A_2, A_3 in $[0, 1]$ are set. These values can be chosen randomly, but in this work, they are decreased down to zero with the progress of the SA. This allows to explore the state space when the temperature is high and focus on a narrower neighborhood when it is low;
- Three raised cosine functions are generated (see Figure 4.14):
  - $\mathrm{rcos}_1(x) = \mathrm{sgn}_1 \cdot A_1 \cdot \frac{\alpha}{2} \cdot \mathrm{rcos}(x, \frac{\alpha}{2}, \frac{\alpha}{2})$
  - $\mathrm{rcos}_2(x) = -\mathrm{sgn}_1 \cdot A_2 \cdot \frac{1-\alpha}{2} \cdot \mathrm{rcos}(x, \alpha + \frac{1-\alpha}{2}, \frac{1-\alpha}{2})$
  - $\mathrm{rcos}_3(x) = \mathrm{sgn}_2 \cdot A_3 \cdot \frac{1}{2} \cdot \mathrm{rcos}(x, \frac{1}{2}, \frac{1}{2})$
- A function g is defined as $g(x) = \frac{\mathrm{rcos}_1(x) + \mathrm{rcos}_2(x) + \mathrm{rcos}_3(x)}{2}$ on $[0, 1]$, by setting $\mathrm{rcos}_1 = 0$ on $]\alpha, 1]$ and $\mathrm{rcos}_2 = 0$ on $[0, \alpha[$;
- The coefficients are chosen so that the route to find has the same shape than g, scaled to the distance $[o\ ,\ a]$ (see fig. 4.12).

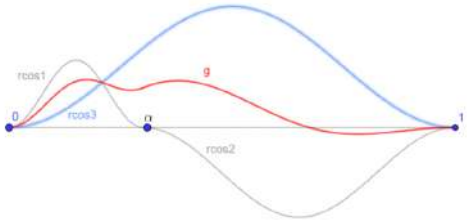


Figure 4.14 – The raised cosine functions to determine the biases in a general case. The function g is the weighted sum of the three others and its shape is the one to give to the path under computation.

4.4 The design of several routes with our algorithm

Once the first route is computed with the method above, a solution can be fully generated by adding the remaining routes, creating a tree structure. The full process for the design of one solution is as follows:

- One tree is constructed for each different runway under consideration, with its layers, vertices and edges. For instance, in the case of an airport using two parallel runways, one for the departures and the other for the arrivals, two trees will be constructed;
- In each tree, the routes are built as follows:
  - The first route (γ_h^0, γ_v^0) is designed with the method explained above;
  - A subset of $\mathscr{L}$ is identified as *merge layers*. They are the layers on which it will be possible for two routes to merge together. These layers cannot be chosen randomly, in order to avoid having two merge points too close to one another. This aspect will be detailed in the next paragraph;
  - The intersection points between γ_h^0 and the merge layers are identified as possible merge points;
  - One of these points is chosen as the starting point for the next route search. Note that in the mathematical modeling, the whole portion of route comprised between the center and the merge point is also counted as part of the new route. Bearing this in mind, all routes in a same tree start at the center of this tree;
  - The second route is designed using the same method;
  - The process continues until all routes have been computed;
- This process is performed for all trees under consideration.

Once a complete solution has been found, it can be optimized with the SA algorithm as described at the beginning of this chapter.

4.4.1 Choosing the merge layers

As stated before, one of the main stakes in our approach is to choose where to merge the routes. We chose to label some of the layers as *merge layers*, as explained in the previous paragraph. In this paragraph, we explain how the merge layers are chosen. We consider here for simplicity purposes that there is only one graph, but in reality, the process is done for each graph.
Firstly, the *center* is always a merge layer. This allows to ensure that each route can be designed, as long as there is at least one edge that ends on their entry/exit vertex (i.e. that there exists an admissible path from the *center* to this vertex). Indeed, the merge layers determine the starting points of all routes. If the center were not considered as a merge layer, a significant part of the possible paths would be discarded, as it would be equivalent to preventing the routes to start on this point. If there is not a single path leading from the *center* to a given ending point of a route, the graph has to be constructed in an other way so that there exists at least one path for each route. The subsequent merge layers are chosen as follows (see fig. 4.15):

- We define the value r as the number of regular layers between two consecutive merge layers. Typically, in our tests, we chose r between 1 and 9 depending on the size of the instance;
- We choose an *offset* value between 1 and r;
- The second merge layer (the first being the center) is $\mathscr{L}_{\text{offset}+2}$;
- Each subsequent merge layer is spaced with the previous one by a gap of r. For instance, the i^{th} merge layer is $\mathscr{L}_{\text{offset}+2+(i-2)\cdot(r+1)}$;
- The number m of merge layer is then

$$m = \underbrace{\lfloor \frac{N_{\mathscr{L}} - (o+2)}{(r+1)} \rfloor}_{\text{nb of regular merge layers}} + \overbrace{2}^{\text{center and offset}} .$$

This method ensures that the merge layers are sufficiently far from each other to abide by the merge constraint 3.5 as long as the space between two consecutive layers is greater than a certain value. However, this process alone cannot guarantee that an entry or exit point isn't located on a merge layer. This situation can be very problematic since it would allow aircraft to enter the TMA directly on another route, which in turn could generate conflicts. In order to avoid such a situation, when an entry/exit point is located on a merge layer, or on a layer that is too close, the merge layer is moved to the nearest layer that is farther than the minimum distance to keep between two merge layers from the layer containing the entry/exit point, as long as it is not too close to another merge layer (see fig. 4.16). If the minimum distances cannot be kept, the merge layer is simply deleted, and becomes a regular layer (see fig. 4.17). Typically, as the entry/exit point that is the farthest from the center is located on the last layer, the last merge layer is often deleted.

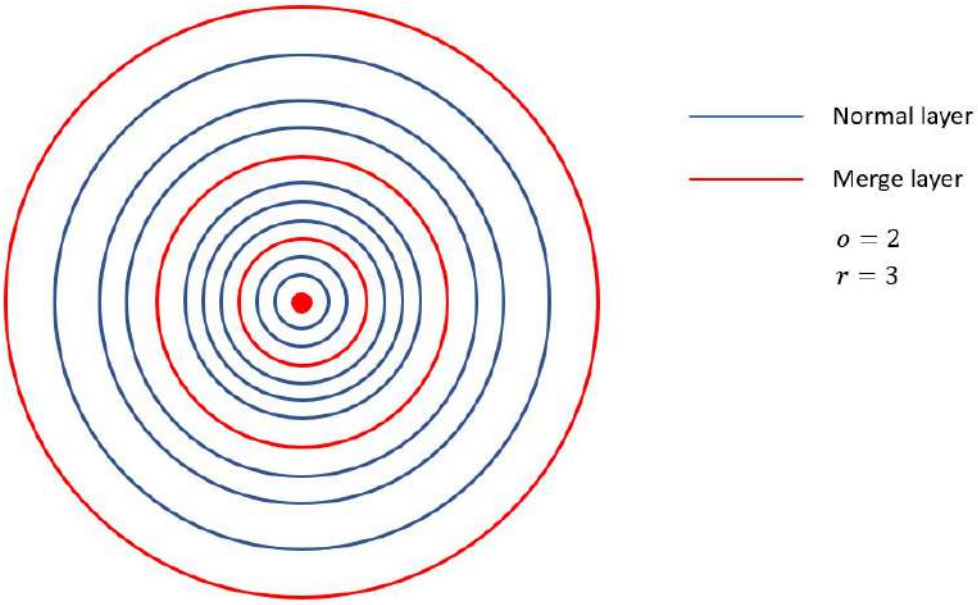
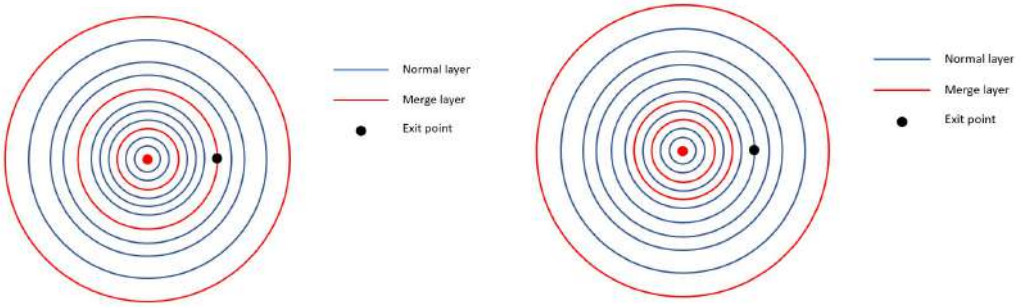


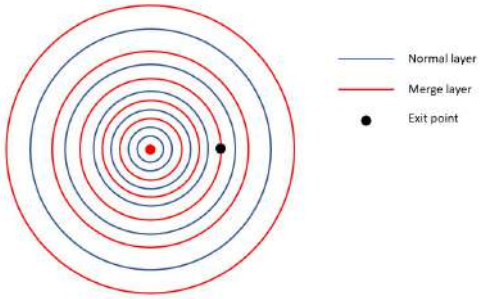
Figure 4.15 – The positioning of the merge layers. In this case, the offset is 2 and r is 3. Apart from the first and second ones, the merge layers are spaced by r regular layers.



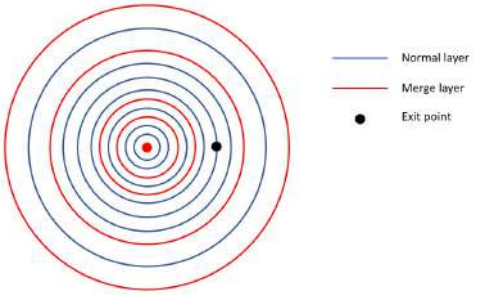
(a) The exit point is located on a merge layer, which is not acceptable.

(b) The merge layer is changed to be on a regular layer that is sufficiently far from both the exit point and the nearest merge layer.

Figure 4.16 – The moving process of a merge layer when it is too close to an exit point.



(a) The exit point is located on a merge layer, which is not acceptable.



(b) The merge layer cannot be moved to a regular layer that is sufficiently far from the closest merge layer. It is therefore deleted.

Figure 4.17 – The deleting process of a merge layer when it is too close to an exit point and it cannot be moved.

4.4.2 Generating a neighbor decision in the SA

When operating the SA, one of the most important processes is the generation of a neighbor for a given solution. A neighbor is a solution that slightly differs from the original solution. In our work, and since the number of iterations in the cooling process is known in advance, we define the value $p \in [0, 1]$ as the *progression* of the algorithm. It is computed as $p = \frac{T - T_{\text{end}}}{T_{\text{init}} - T_{\text{end}}}$ with T the current temperature. We also define a random value $R \in [0, 1]$ that is drawn at every neighbor generation. The possibilities to generate a given type of neighbor are set as explained by Algorithm 12 and illustrated by the flowchart in fig. 4.18. In Algorithm 12, several operations are not detailed, such as the management of the route connections or of the level flights. These are addressed in the next paragraphs. Some of the operations listed in Algorithm 12 change many features in the initial solution. These steps aim at exploring the search space in the beginning of the algorithm. It can be seen that after 60% of progression of the algorithm, it is impossible to fall in these cases. The algorithm then focuses on narrowing the best solutions' neighborhoods.

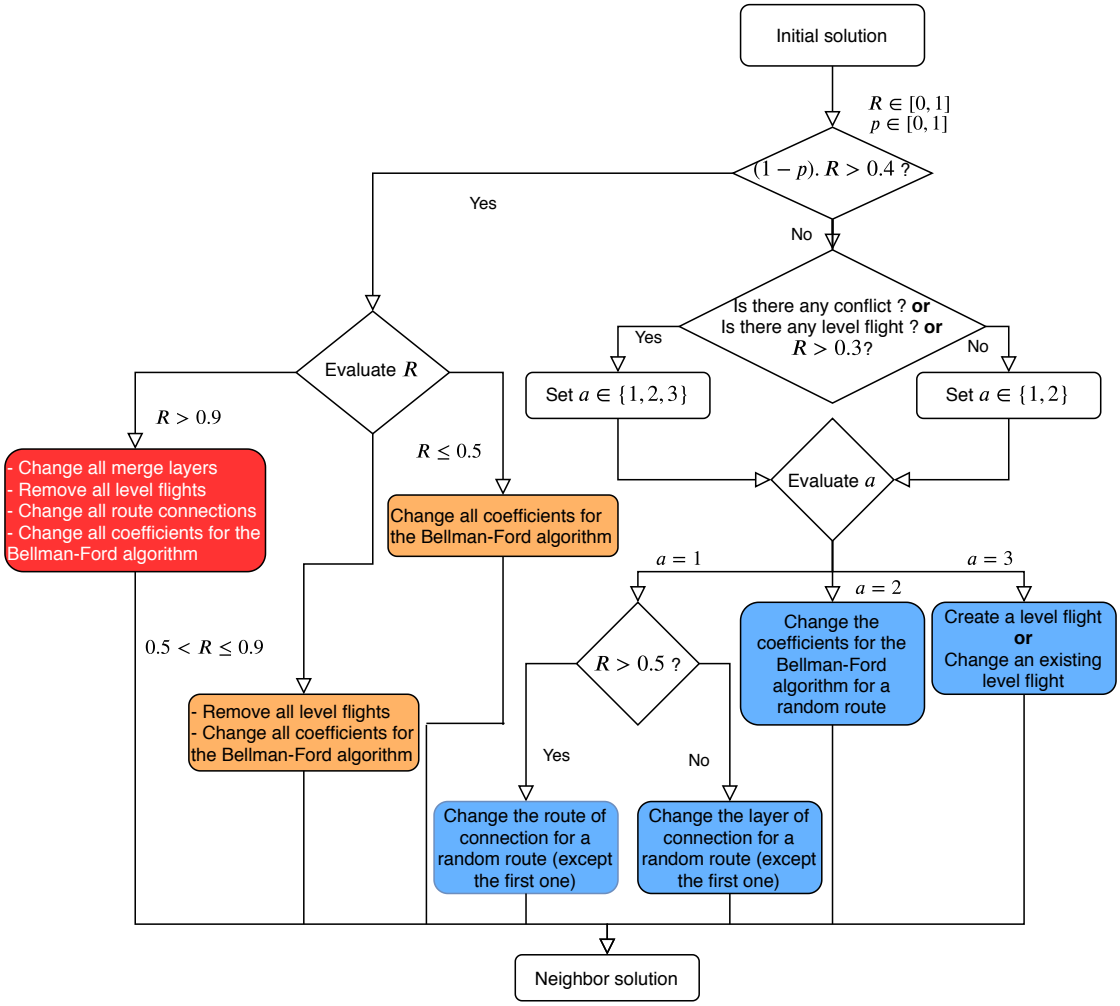


Figure 4.18 – The illustration of Algorithm 12 for the neighbor generation in the SA. The major changes are colored in red, the medium changes in orange and the minor changes in blue. At the beginning of the SA, the major changes are more likely to happen. Past a certain progression of the algorithm, only minor changes can be performed.

Algorithm 12 The neighbor generation in the SA method. Major changes can happen at the beginning of the algorithm. Past a given progression of the algorithm, only minor changes can happen.

Require: the progression p, the random number R, an initial solution S to the problem

Initialization: Create a new solution S' that is a copy of S
if $(1-p) \cdot R > 0.4$ **then**
    if $R > 0.9$ **then**
        Change all merge layers
        Remove all existing level flights
        Change all route connections
        Change all coefficients for the Bellman-Ford algorithm
    else if $R > 0.5$ **then**
        Remove all existing level flights
        Change all route connections
    else
        Change all coefficients for the Bellman-Ford algorithm
    end if
else
    Set a variable $nbChoices = 3$ if there are conflicts (between a route and an obstacle, or between two routes) **or** if there are level flights in S' **or** if $R > 0.3$, 2 otherwise
    Set a (the *action*) a random integer between 1 and $nbChoices$
    if $a = 1$ **then**
        if $R > 0.5$ **then**
            Change the route of connection for a randomly drawn route except the first two
        else
            Change the layer of connection for a randomly drawn route except the first one
        end if
    else if $a = 2$ **then**
        Change the coefficients for the Bellman-Ford algorithm for a randomly drawn route
    else
        Change the level flights for a route on which there is a level flight, or a route that is in conflict with an obstacle or another route, or create a level flight on a randomly drawn route
    end if
end if

4.4.3 Changing the level flights

In our work, we decided to include the management of level flights in order to be able to tackle complex scenarios where two routes can cross in the horizontal plane. As stated in chapter 3, there are three constraints that apply to the level flights:

- There is a limit to the number of level flights that can appear in the solution.
- A level flight cannot be too short in terms of horizontal distance: it wouldn't make sense in a real operational context.
- A level flight cannot be too long in terms of horizontal distance: the aircraft has to climb enough to reach the en-route sector.

In order to enforce these requirements, we decided to impose that a level flight is always applied on the whole part of the horizontal profile of a route that is comprised between two merge layers (see fig. 4.19). In doing so, the second point is always considered valid. The management of the first requirement, concerning the number of level flights, is achieved with the use of a table shared between all sets of routes in a solution, whose size is the maximum authorized number of level flights. It holds information about each level flight by the means of the tree number, route number and layer number to which it belongs (see fig. 4.19). At the beginning of the

Level flight number	Route	Layer
1	1	2
2	1	4
3	/	/
4	/	/

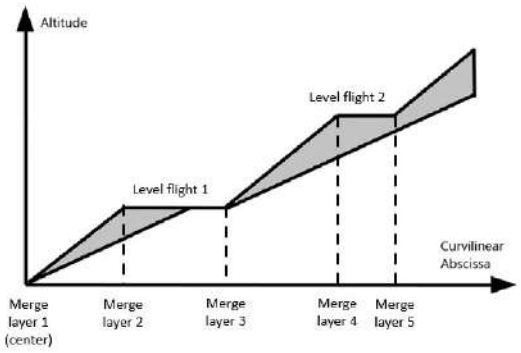


Figure 4.19 – The illustration of the management of the level flights in our approach. The table holds for each authorized level flight the route and the merge layer on which it is applied.

algorithm, this table is empty. Once it contains at least one level flight, the possible actions concerning the level flights in Algorithm 12 are as follows (see fig. 4.20):

- Changing an existing level flight (i.e. changing the graph, route, and/or layer of an existing level flight);
- Removing an existing level flight;
- Creating a new level flight. If the maximum number of level flights is already reached, the new one replaces an existing one, chosen at random.

If there is no existing level flight, the only option is to create one.

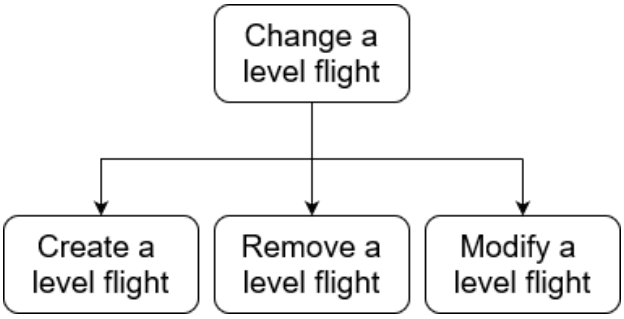


Figure 4.20 – The possible choices for a change in the level flights.

4.4.4 Changing the connection of a route

One of the main points addressed by our algorithm is the management of the route connections. The way two routes merge together has already been described in chapter 3. In this paragraph, we give in detail the way in which the connections are created and modified throughout the algorithm's execution. For a given tree, the connections are handled by the means of a dedicated table whose size is the number of routes (see fig. 4.21). The index i corresponds to the connection information of route i, which is given by the number of the route on which route i is connected, and the number of the layer on which the connection takes place. When operating the SA, the table is managed in the following way:

- The first route doesn't connect on anything, so we set the information to a default value, like (-1,-1).
- All connections are decided before the first route is designed.
- When the route i is created or modified, the algorithm checks the table between indices $i+1$ and the table size. Each route that was connected to route i is to be computed again, since their current connection may not be acceptable anymore (it could induce a violation of the turn angle constraint, or the obstacle avoidance constraint, for instance) (see fig. 4.22). The other indices are not checked, since a route i can only connect to a route between 1 and $i-1$. Therefore, none of these routes are connected to route i.

Route number	Route of connection	Layer of connection
1	/	/
2	1	3
3	2	4
4	1	1

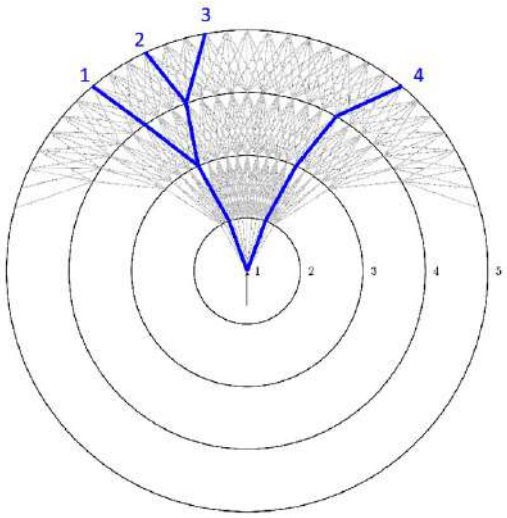
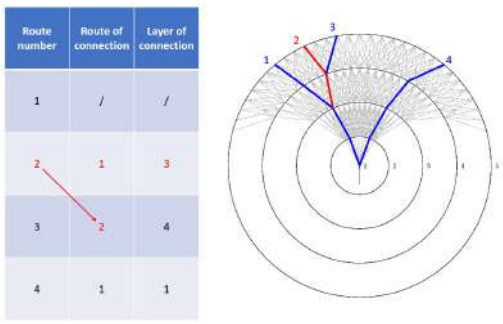


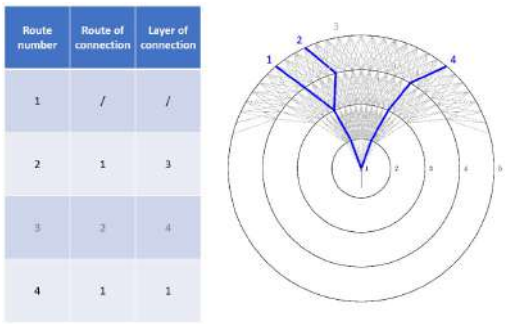
Figure 4.21 – The illustration of the management of the route connections in our approach. The table holds for each route the route and merge layer on which the connection must be done.

- When creating or modifying a connection, the algorithm checks whether the connection it tries to create already exists. If this is the case, it proceeds as follows:
  - It tries to establish a connection on a different layer of the chosen route;
  - When no solution is found, it tries to connect on a random route with an index lower than that of the route to connect;
  - The last point is repeated until an acceptable connection is found, or a certain arbitrary count is reached. We set the count limit at 100;
  - If the count limit has been reached, the route connects at the center by default, even if another connection exists. This is the only way to ensure that the route can be designed by the algorithm. This kind of degraded cases can happen when the number of merge circles is too small, or if the first routes to design are too short. In these cases, there are too few connection possibilities and they become saturated (see fig. 4.23). The best way to avoid this situation is to include more merge layers whenever possible.

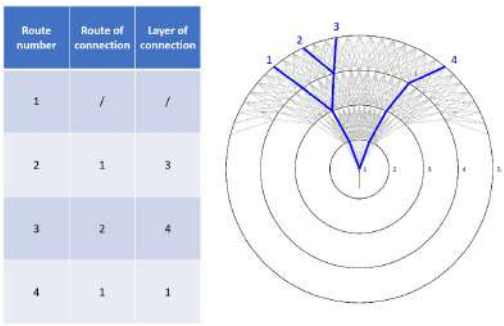
- It can happen that during the execution of the algorithm, a connection leads to an infeasible design (see the example of fig. 4.24). In that case, the impossibility is detected by the modified Bellman-Ford algorithm, as the arrival point is never *marked*, which means that there is no path between the connection point and the arrival point. When this happens, the algorithm changes the connection to the nearest merge



(a) Route 2 must be recomputed.



(b) Route 2 is recomputed. Since route 3 is connected to route 2, it must be recomputed as well.



(c) Route 3 is recomputed. The other routes are not checked for recomputation as they are not connected to route 2.

Figure 4.22 – The illustration of the propagation of the change in one route in our method. The changes propagate to all subsequent routes that are connected to the original route.

layer in the direction of the center, and repeats this process until a feasible route can be generated (see fig. 4.25). In some cases, the situation is similar to the one described in the last point, and the route connects on the center regardless of other existing connections.

Route number	Route of connection	Layer of connection
1	/	/
2	1	1
3	x	x

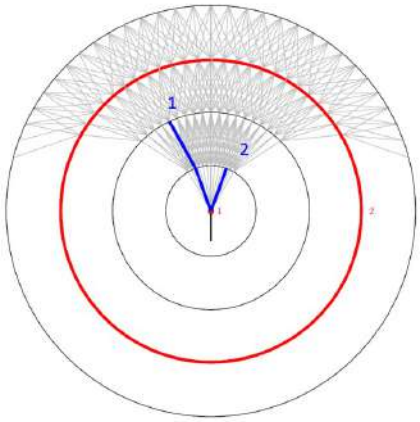


Figure 4.23 – Route 3 cannot be computed, since it cannot be connected anywhere: all possible connection points are taken (the merge layers are displayed in red, the regular layers in black). The only possible connection point (route 1, layer 1) is already taken by route 2.

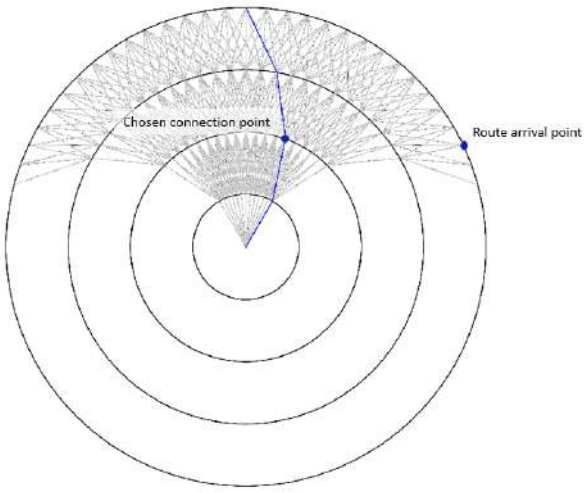
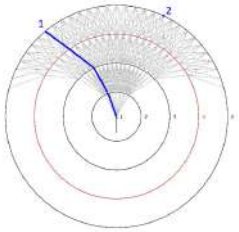


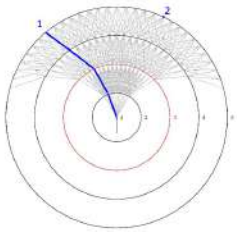
Figure 4.24 – An example of connection for which no solution can be found. There exists no path between the selected merging point and the route arrival point.

Route number	Route of connection	Layer of connection
1	/	/
2	1	4



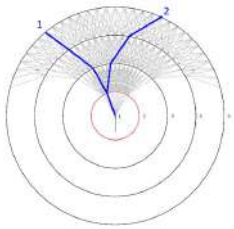
(a) Route 2 must connect to route 1 on layer 4, which is impossible.

Route number	Route of connection	Layer of connection
1	/	/
2	1	3



(b) The connection layer for route 2 is lowered to 3. There is still no possible path for this connection.

Route number	Route of connection	Layer of connection
1	/	/
2	1	2



(c) The connection layer for route 2 is lowered to 2. A route can then be found for this connection.

Figure 4.25 – No route can be found in the graph for route 2 with the given connection. The merge layer is decreased until a route can be found. Only the merge layers are displayed.

4.5 Solution evaluation

In the simulated annealing meta-heuristic, it is necessary to be able to evaluate frequently a given solution (i.e. a set of routes). This evaluation is carried out by the means of a grid with the following features:

- The grid covers at least the area covered by the trees.
- Each cell of the grid is a square whose side length is not greater than the minimum horizontal separation (for example, their side is 1NM-long in this work).
- Each cell holds the following information:
  - The height of the highest ground obstacle that can be found in this cell (mountain, antenna, building...);
  - The minimum and maximum altitudes of a forbidden flight zone in this area (if any);
  - The density of population on the ground (if any).

Thus, the obstacles are 'widened' due to the discretization. This allows to take additional margins, but could also lead to the loss of potential solutions if the grid squares are too large. This is why it is important to keep them rather small.
In order to simplify the path search process, some of the constraints presented in 3 are relaxed. They are the following constraints:

- maximum turn angle constraint;
- route separation constraint;
- obstacle avoidance constraint.

Instead of looking for routes that abide by these constraints in the path search, we authorize the algorithm to design any set of routes. However, during the evaluation process, whenever a constraint is violated, the objective function is heavily penalized, which forces the algorithm to discard such solutions. Note that in some complex cases, the algorithm may terminate with its best solution still violating some constraints. In this case, one or several other tries are made. If the algorithm still cannot find an admissible solution, either the discretization or the limiting parameters (minimum/maximum slopes γ, maximum turn angle...) must be changed.
To carry out the evaluation phase, each route is discretized with an arbitrary step, for example d_h (see fig. 4.26). Each of the created points belongs to one cell of the grid according to its coordinates in the plane, and is given a cost value, which is initialized at 0. The evaluation is done by processing successively each cell of the grid containing at least one point. We also define the *direct predecessors* and *direct successors* of a point P with a given curvilinear abscissa $\gamma(P)$ as all the points of the same route as P whose curvilinear abscissas are respectively within $[\gamma(P) - 2d_h, \gamma(P)]$ and $[\gamma(P), \gamma(P) + 2d_h]$ (see fig. 4.26). These values have been arbitrarily chosen,

so that a curved path is not penalized, but a zigzagging one should be.

When a cell contains at least one point P, its evaluation is carried out as

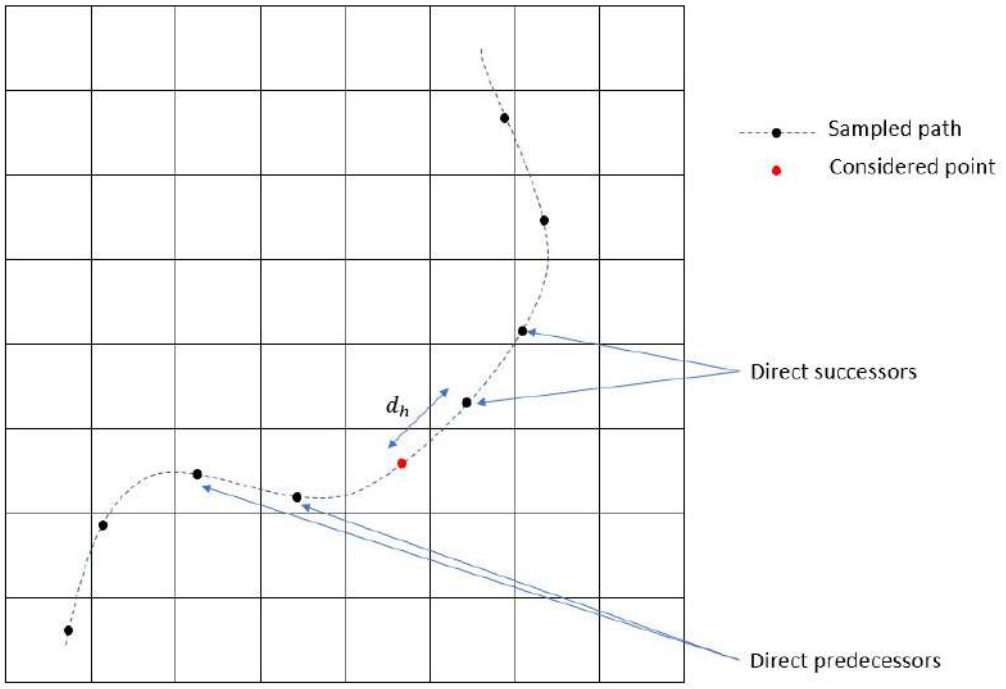


Figure 4.26 – The discretization of one route for the evaluation process. The direct predecessors and direct successors are identified.

follows:

- The cost of P is increased by the value of the discretization step, so as to take into account the route length objective;
- The cell containing P is checked for the presence of a city. When there is one, the formula given in chapter 3 is applied, and the result is added to the cost of P;
- All cells in a d_h radius around P are checked for obstacle intersections. For each cell checked, and each existing obstacle within them, if the obstacle separation constraint 3.1 is violated, the cost of P is dramatically increased;
- All cells in a d_h radius around P are checked for other routes. For each cell checked, and each existing route point P' within them, the following tests are carried out (see the illustration of each case in fig. 4.28). If P' is at a distance (in 2D) less than or equal to d_h and:
  - belongs to a route that is in another graph, or
  - belongs to a route in the same graph that is not involved in a connection with the route to which belongs P, or
  - belongs to a route that is involved in a connection with the route to which belongs P, but neither one of the direct predecessors of

P is a direct predecessor of P', nor one of the two points P and P' is a direct predecessor of the other, or

- belongs to the same route as P but is neither one of its direct predecessors or successors,

and the vertical distance between P and P' is less than d_v, then the cost of P is dramatically increased. If one of the above conditions are met, but the vertical distance between P and P' is more than or equal to d_v, a slight increase in the objective function is applied, so that the algorithm favors planar graphs. This choice has been made to keep the cognitive load of the controllers at the lowest possible level, since a crossing between two routes always draws attention, even when they are separated in the vertical dimension.

On top of this cell-by-cell evaluation, the maximum turn constraint 3.2 is also checked, by considering successively each individual route. Starting from the center and going towards each entry/exit point, the algorithm checks each triplet of successive points. If the constraint is violated, the cost of the middle point is dramatically increased. See fig. 4.27 for the identification of the cells to check, and fig. 4.28 for an illustration of the possible cases that can occur when another route point is found in the neighboring cells. Note that in our work and with our modeling, the case

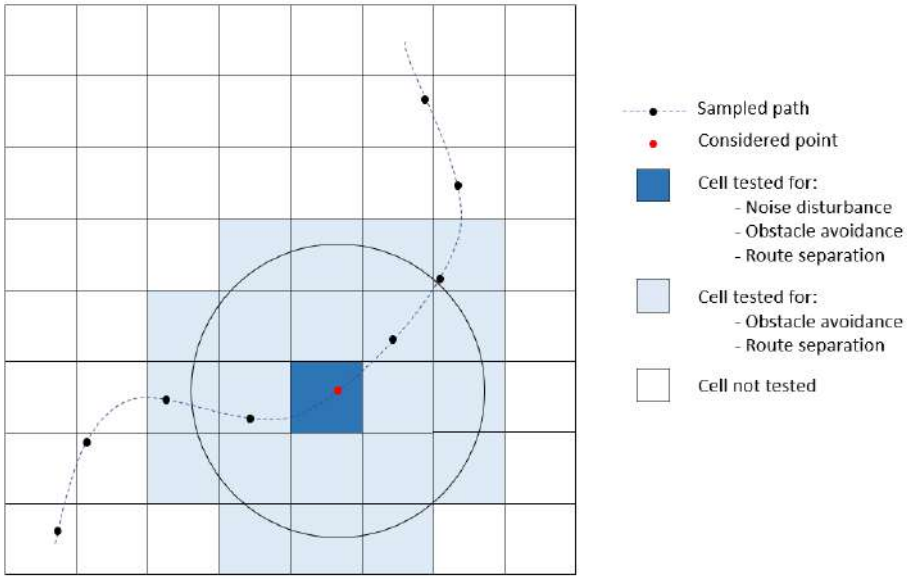
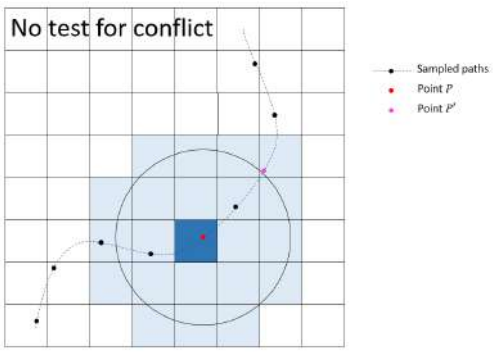
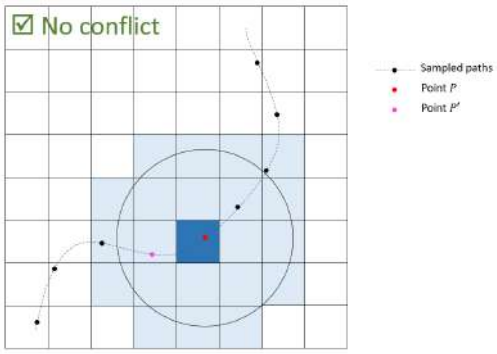


Figure 4.27 – The evaluation process for one point of one route. One cell is tested for all criteria. Neighboring cells in a given radius are tested for all criteria but noise disturbance.

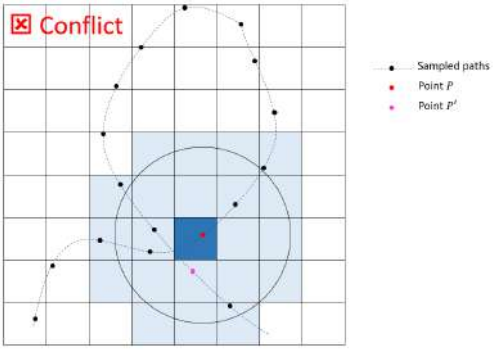
where P and P' are in conflict while belonging to the same route is very unlikely to happen, since an edge can only go from a layer i to a layer $i+1$. It may however still happen, with an adequate choice of layers and a wide maximum turn angle.



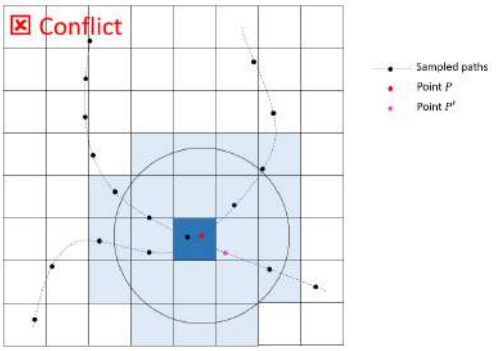
(a) P' is not checked for conflict.



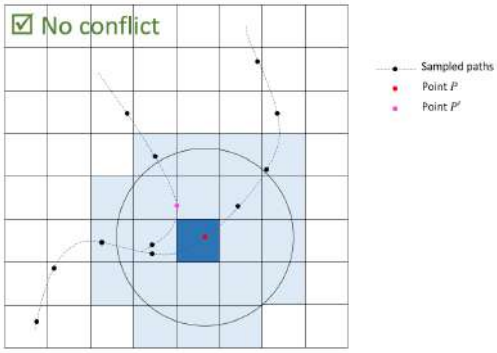
(b) P' is a direct predecessor of P.



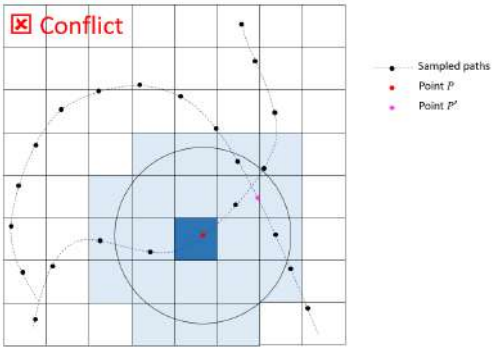
(c) P' is not a direct successor of P.



(d) P' belongs to another route.



(e) P' and P have common direct predecessors.



(f) P' and P don't have any common direct predecessor.

Figure 4.28 – The detection of conflicts between routes in our algorithm. Each possible case for the nature of P' is illustrated.

In this chapter, we presented two resolution approaches for the optimal SID/STAR design problem. The first one is inspired by the principle of dynamic programming. However, resolving the problem as it is formulated with an exact algorithm is not feasible, as we established that both space and time complexities are exponential. Therefore, the method features a heuristic in order to reduce the risks of dead ends when a solution exists. Nevertheless, this first approach has major drawbacks, such as the necessity to impose an order for the route generation, or the fact that it cannot find any solution when there might be one in specific conditions. In order to avoid these problems, a second method is introduced, based on the Simulated Annealing metaheuristic. This method allows to explore the various possibilities of solutions while being less prone to getting stuck in local minima. However, it also requires to tune a certain number of parameters, often empirically. In the next chapter, we present the results that were obtained for both approaches on an artificial test case. We also measure the performances of the SA approach on three other instances depicting real-life scenarios, taken from the literature. A comparison between the results of our approach and the ones from the literature is provided.

Chapter 5

Simulation results

In this chapter, we present the results obtained with our methods. These tests were performed on various instances. First, we tested the approach based on the principle of dynamic programming, on an artificial instance. This test allowed to highlight the strengths, but also the limitations of this approach. Then, we tested the SA approach on the same instance. Given the comparative results on the performances for both methods, we then chose to continue the tests with only the SA approach. The subsequent tests were performed on three instances: Stockholm, Paris Charles-de-Gaulle and Zurich. For each one of them, several configurations were tested (for instance by varying the number and shape of the layers).

5.1 Experimental setup

We start here by laying out the experimental conditions in which the tests were performed. A short introduction of each test case is provided, then some specificities about the way to measure the results are given. Finally, we give the numerical values used to parametrize the SA algorithm. All the tests were run on a 2.70 GHz Intel Core i7 processor with 16GB of RAM on a Windows operating system, with a code written in java.

5.1.1 Introduction of the test cases

5.1.1.1 The artificial instance

The first instance on which the algorithm was tested is an artificial instance, designed specifically for the method presented here. Two resolution methods are tested on this instance: the dynamic programming based heuristic, and the SA-based method. For the first approach, we successively consider a forbidden zone and a mountain. Then, we present a case with a mountain, a forbidden zone and a city together with six routes to design. These three obstacles are then considered independently for the SA-based approach, then all together again for the same six routes to be designed.

5.1.1.2 The Stockholm instance

The second test case for our algorithm is taken from the literature [114] and consists in designing five STARs in the vicinity of Sotckholm Arlanda's airport. It is the first test case taken from a real-life scenario. In this case, the only obstacles to consider are the cities in the area. A comparison between our results and those from [114] is provided.

5.1.1.3 The Paris Charles-de-Gaulle instance

The third scenario on which our method was tested is the Paris Charles-de-Gaulle case. For this design, we took the input data from [45] in order to be able to compare our approach with a state-of-the-art method. The difficulty in this specific case is to be able to design a large number of routes.

5.1.1.4 The Zurich instance

The last test case for our method is also taken from [45]. It represents the vicinity of the Zurich airport with nine routes to be designed. The main difficulty of this instance is to be able to take high mountains into account in the design, in a TMA of a relatively small size. The results are also compared with those from [45].

5.1.2 Measuring the test results

The tests presented in this section were all run at least 30 times each for the SA-based approach, in order to measure mean values for the various criteria. In the tests run for this work, some of them failed, in the way that the solution provided by the algorithm violated a *conflict constraint* (i.e. at least one route passes through a ground obstacle, a forbidden zone or another route). In this case, the algorithm is run again, and so until the solution is feasible (i.e. it doesn't violate a conflict constraint). The 30 to 50 tests mentioned above represent only the number of successful tests. When giving the numerical results of our experiments, the computation time will be mentioned. This computation time always takes into account the failed tests. Thus, when multiple solutions are rejected, the computation time is lengthened. Another feature mentioned in the results will be the proportion of failed tests. These are always counted against the total number of tests, which is the sum of successful and failed tests. Therefore, the proportion of failed tests is always comprised between 0 and 100%. Therefore, for instance, a proportion of failed tests of 50% on a series of tests means that there were as many failed tests as successful ones, since for each test, the algorithm starts over every time the solution is not feasible.

5.1.3 The parameters used for the SA

The performed tests all make use of the Simulated Annealing, with the same parameters. These parameters have been tuned, in preliminary tests, in such a way that they fit the method. We set these parameters to the following values:

- The initial temperature is chosen by iteratively generating a solution and a neighbor (in our case, 100 times) and measuring the average acceptation rate of the neighbor against the initial solution with equation 4.3. If this rate is lower than 80%, the temperature is increased by 50% and the process starts over. The starting temperature is set to 67.5;
- The stopping criterion is by default the temperature reaching the initial temperature divided by 10,000;
- In the cooling phase, the temperature is decreased by 10% of its current value each time it is decreased. This induces that the stopping criterion is equivalent to reaching a certain number of iterations (in our case, 88);
- The number of neighbors kept for each vertex expansion (as described in section 4.3.1 by fig. 4.11b) is 10;
- The coefficients for the objective function are the same for all test cases: $\alpha = 2/29$, $\beta = 25/29$ and $\gamma = 2/29$.

5.2 The artificial instance

For our first test scenario, we decided to create an artificial layout especially designed to test all the possibilities of the algorithm: taking cities, ground obstacles and military zones into account. Firstly, we apply the dynamic programming algorithm on three separate cases. Two of them consist in designing a single route, respectively with a forbidden zone to be flown under with a level flight, and a mountain very close to the runway, to be flown over. The third test case for this method involves a forbidden zone, a mountain and a city, with six routes to be designed. We then present the results obtained for the SA-based approach. In these tests, we successively measured the behavior of the algorithm on each of the three obstacles used for the last test of the dynamic programming method individually. Then, the performances of the SA-based approach are measured on the same case than before, with all obstacles and six routes to design at the same time. Finally, we added a second runway in the scenario. In all tests conducted on the artificial instance, all routes are considered to be STARs. The layout for the obstacles is displayed in fig. 5.1, and the corresponding numerical values are given in Table 5.1. In Table 5.1, each obstacle is given in the form $(((x_1, y_1); (x_2, y_2); \ldots; (x_n, y_n)), l_o, u_o)$ and the city in the form $(((x_1, y_1); (x_2, y_2); \ldots; (x_n, y_n)), \eta(x, y))$ where:

- $(x_1, y_1); (x_2, y_2); \ldots; (x_n, y_n)$ are the coordinates of the base polygon giving the location of the city
- l_o is the lower altitude of the obstacle (0 being on the ground)
- u_o is the upper altitude of the obstacle
- $\eta(x, y)$ is a function indicating the population on the ground at the point (x, y).

The data used for the routes is presented in Table 5.2.

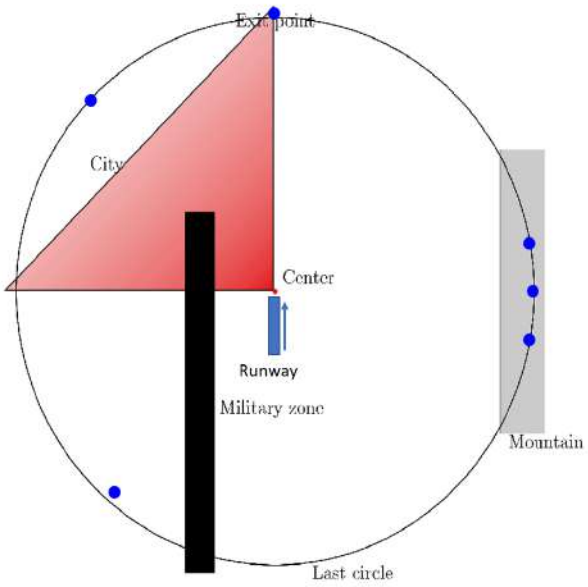


Figure 5.1 – The layout used to test the algorithm in the SA-based approach.

Obstacle number	Obstacle type	(B, l, u) or (B, η)
1	Mountain	$\Big(\big((300,-180);(300,178);(359,178);(359,-180)\big),\ 0,\ 12\Big)$
2	City	$\Big(\big((-2,1);(-360,1);(-2,359)\big),\ (x,y) \mapsto 806 - (y - x)\Big)$
3	Military Area	$\Big(\big((-120,-355);(-80,-355);(-80,100);(-80,-355)\big),\ 15,\ 50\Big)$
4	Close mountain	$\Big(\big((50,-180);(50,178);(79,178);(79,-180)\big),\ 0,\ 12\Big)$

Table 5.1 – The obstacles and city for the artificial test case (see Figure 5.1).

Route number	Traffic flow	Graph center	Entry point (polar coordinates)
1	26.67%	(0,0,0)	(346NM, 0°)
2	24%		(346NM, 10°)
3	21.33%		(346NM ,350°)
4	13.33%		(346NM ,90°)
5	12%		(328.5NM, 135°)
6	2.67%		(328.5NM, 230°)

Table 5.2 – The data used for the routes in the artificial instance (1 runway).

Additionally, the data relative to the constraints in this case is the following:

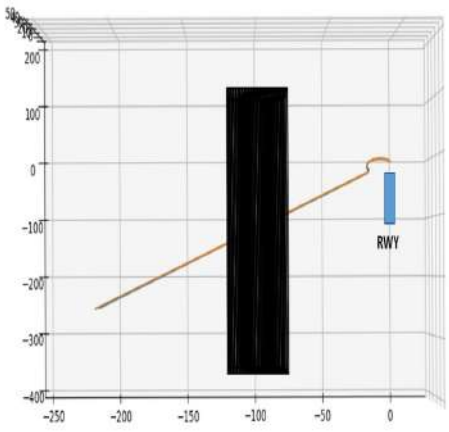
- The runway is oriented by the vector $(0, 1)$
- The altitude range of the exit points is not relevant (so all ranges are accepted in the solution)
- Maximum turn angle $\theta_{max} = 30°$. It is the maximum authorized angle between two consecutive arcs in the horizontal profile.
- Minimum angle at merge points $\theta_{min} = 15°$
- Minimum slope α_{min}: 3.24%
- Maximum slope α_{max}: 16.2% Note that the values for the slopes are taken from the standards communicated by the ICAO.
- The maximum number of level flights $n_{max}^{LF} = 4$
- The minimum length of the level flights was set to $l_{min}^{LF} = 10NM$ and the maximum length l_{max}^{LF} is not relevant here
- There is no orientation imposed on the entry points

For all tests conducted on the artificial instance, and for the ones presented in the rest of this document, the layers have been arbitrarily chosen. The layers must provide a discretization both **precise** enough to represent the TMA, as if there are not enough, a significant part of the possible solutions can be missed, and **sparse** enough to keep the routes close to real ones, with a small enough number of possible turns, and a minimum distance between two potential decision-making points. It also keeps the instance within the computational capabilities of the algorithm, as the computation time increases strongly with the number of points used to discretize the TMA.

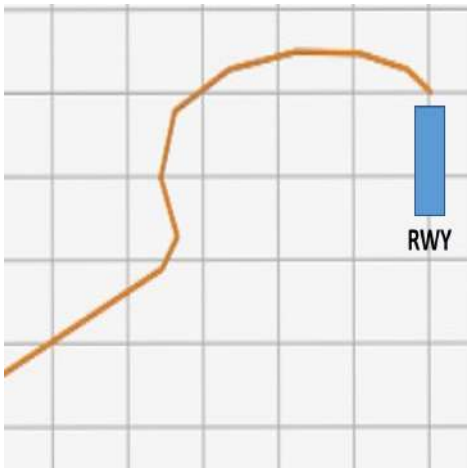
5.2.1 The dynamic programming based approach

5.2.1.1 Designing one route

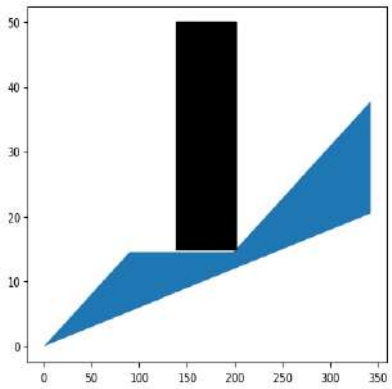
We first tested the dynamic programming approach with two situations. The first one is the design of one route (route 6 from Table 5.2) with a forbidden zone to avoid (obstacle 3 from Table 5.1). We chose to use circular layers for all tests in this section, with 100 layers and 360 points equally distributed on each layer (it is the same discretization as the "heavy discretization" used in the first test case of the SA approach, described in the next section). The result is shown in fig. 5.2, with both the horizontal and the vertical profile of the route. It can be seen that the route avoids



(a) The horizontal profile of the route with a forbidden zone to avoid.



(b) A more precise view of a portion of the route. The turns are made as the discretization allows. The maximum turn angle constraint is met.

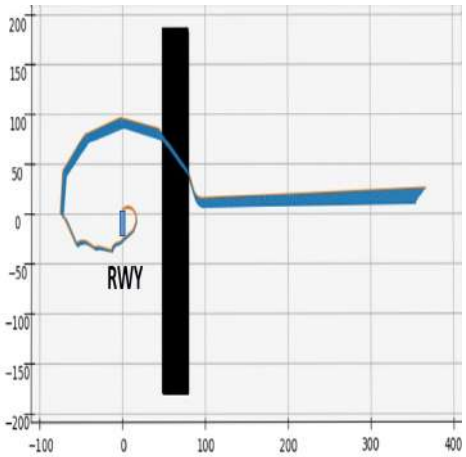


(c) The vertical profile of the route.

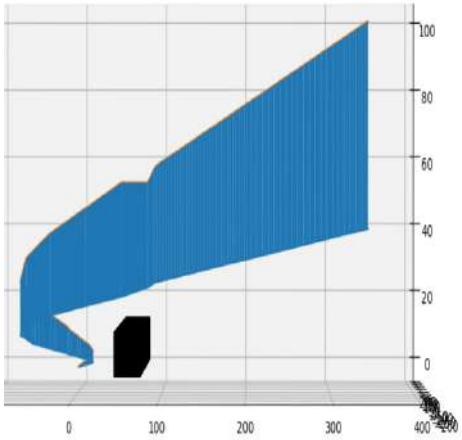
Figure 5.2 – The dynamic programming approach with one route to be designed with a forbidden zone to avoid.

the forbidden zone by making use of a level flight, of minimum possible length. The route is as straightforward as possible between the center and

the exit point of the TMA.
The second test that was conducted aims at designing a route that passes over a mountain that is located very close to the runway. Therefore, the route needs to be longer before crossing the mountain, in order to gain altitude. The results are shown in fig. 5.3. The results show that the



(a) The horizontal profile of the route with a mountain to avoid.



(b) The vertical evolution of the route.

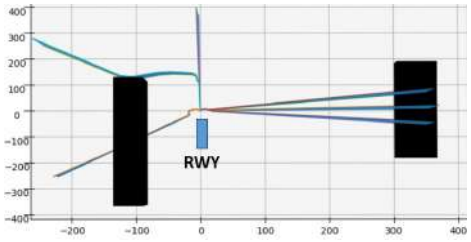
Figure 5.3 – The dynamic programming approach with one route to design with a mountain to avoid.

algorithm is able to choose the shortest route among only the feasible ones, thanks to the preprocessing phase indicating the minimum altitude required to pass the obstacles. Therefore, this approach is very efficient to design a single route with any kind of obstacle. In both cases presented above, the method takes between 6 and 7 min to yield the result. The irregularities that can be observed are due to the choice of the discretization. They could be removed, for instance, with a smoothing technique applied in a postprocessing phase, which is not the case here.

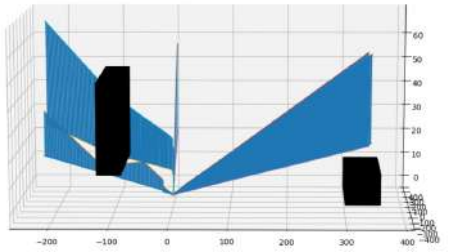
5.2.1.2 Designing several routes

In the next test case, we decided to design six routes (see Table 5.2) with three types of obstacle: a city, a forbidden zone and a mountain (obstacles 1, 2 and 3 from Table 5.1). For this test, the routes are designed sequentially, two times each (cf Algorithm 7). In the first design, each route is computed with the preprocessing method (cf Algorithm 5), without taking the other routes into account. Therefore, the result is for each entry point the shortest route from the center. When all routes have been computed in this fashion, the pairwise distance between them is measured. Then, for each route except for the first one, a new starting point is chosen, as explained and illustrated in section 4.4, as the last point from another route that was too close to the route under consideration. The results are shown in fig. 5.4.

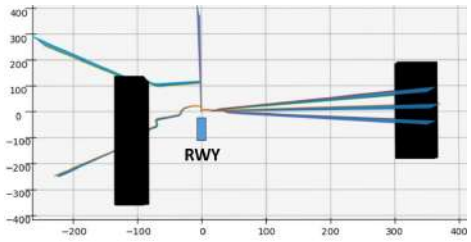
The results were obtained in approximately 43 min, which is a very



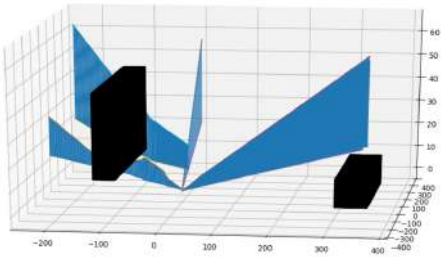
(a) The six routes at the end of the first computation, without connection management (horizontal profiles). All routes connect on the same point: the center.



(b) The vertical profiles at the end of the first computation.



(c) The six routes at the end of the second computation, with the merge separation constraint taken into account (horizontal profiles). Each route connects on a different point so as to spread out the merging points. Each route is connected on the first available point.



(d) The Vertical profiles at the end of the second computation.

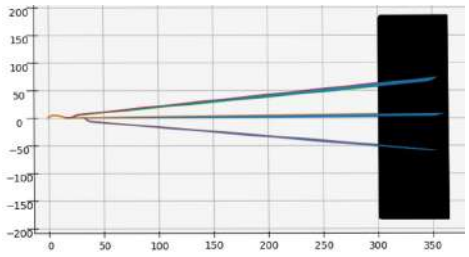
Figure 5.4 – The result obtained with the dynamic programming approach for six routes and three obstacles, with a management of the connection points for the routes.

acceptable computation time considering the strategic context of the study, and show that the algorithm is capable of taking all constraints into account. However, the connection points for the routes have to be chosen either manually, or with a heuristic. The same limitation occurs when several graphs are to be taken into account, as the order for the design has to be chosen beforehand.

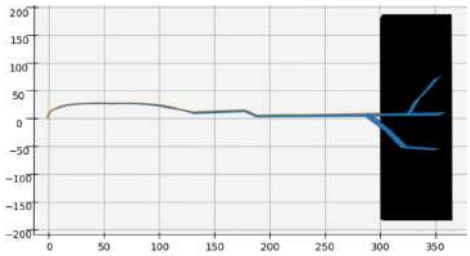
As a comparison, we ran the same test with the dynamic programming-based heuristic and with the Simulated Annealing approach, with routes 1, 2 and 3 to design (see Table 5.2) with the mountain to avoid (obstacle 1, see fig. 5.1 and Table 5.1). The numerical results for this test are provided in Table 5.3, and the shapes of the routes are illustrated in fig. 5.5. The results show that the dynamic programming-based approach is slower to find a solution than the SA. We can also note that the shape of the routes is more appropriate for the SA approach, as the routes merge as soon as possible, far from the center. This allows for the controllers to have mainly one route to monitor instead of three, which is valuable. As can be seen from these results, the SA approach is more likely to yield satisfactory results in the context of this

Measurement	DP-based heuristic	SA approach
Computation time	20 min 34s	18 min 11s
Route length	1013.89	1280.79
Graph weight	993.73	700.97

Table 5.3 – The mean results for the test of comparison between the dynamic programming-based approach and the SA approach



(a) The three routes designed by the dynamic programming-based heuristic. The merging points of the routes are close to one another, and close to the center.



(b) The three routes designed by the SA approach. The merging points of the routes are close to one another, but far from the center. The graph weight criterion is thus better than for the dynamic programming-based heuristic.

Figure 5.5 – The comparative results for the design of three routes over a mountain by the dynamic programming-based heuristic and the Simulated Annealing approach. The SA allows for a better "compactness" of the solution, while the dynamic programming-based heuristic favors direct routes.

thesis. Therefore, we analyze this approach in more detail over the next section, before testing it against cases taken from the literature.
In conclusion, the dynamic programming-based approach has many advantages for the design of a single route: the result is easily reproducible ; the method doesn't need parametrization and the input is very simple (only the starting and ending points are required, besides the operational features, such as the climb slopes). However, on its own, it lacks the exploration power provided by metaheuristics, which is necessary to tackle situations with more routes to handle, and with the introduction of a heuristic, it is not guaranteed to find a solution, even when there might be one. In the next section, we present the results obtained for the same instance, with the SA-based approach.

5.2.2 The Simulated Annealing based approach

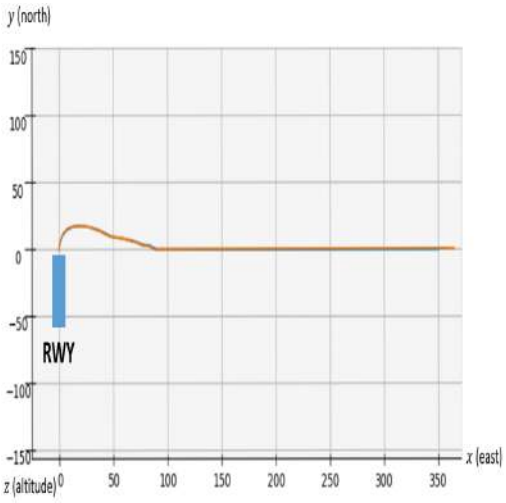
For the SA-based approach, we first tested the case where only one runway is taken into account, then we added a second runway.

5.2.2.1 One runway

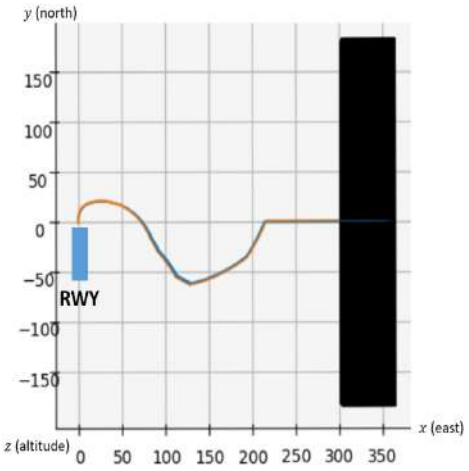
In the case where only one runway is taken into account, we start by designing only one route at a time, in order to test the algorithm on each type of obstacle. Then, several routes are computed together, with all obstacles at the same time. In all this section, all tests were successful. Therefore, the proportion of failures is 0% for all cases with a single runway.

5.2.2.1.1 One route design and one obstacle

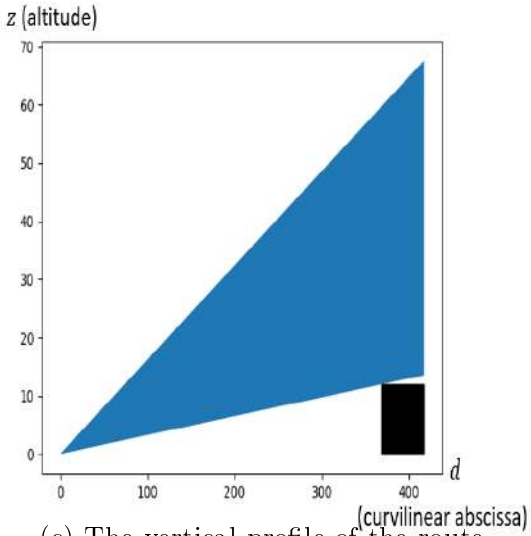
This test case successively considers the effects of a mountain, a city and a military zone on the design of one route. For each of these tests, the algorithm was first run without any obstacle, so as to set a reference. In all this section, the routes have been designed only once, as there was no reason to measure average values for the route length or the computation time. The aim here is to observe the behavior of the algorithm when a route must deviate from a straight line to avoid an obstacle. The type of layers used here is circular. The first obstacle that has been studied is a mountain, on the right-hand side of the map (see Figure 5.1), too high for the route to fly in a straight line from the center to its exit point. The results are shown in Figure 5.6. It can be seen from Figure 5.6 that the route is lengthened for the aircraft to gain altitude and to be able to fly over the obstacle.
The second obstacle that has been studied is a city, as shown in Figure 5.1, with the highest population density at the center and decreasing towards the exterior. As before, the algorithm was first run without any constraint to set a reference. The results are shown in Figure 5.7. The turn towards the exit point is delayed so as to avoid flying over the most populated areas of the city and cause noise disturbance. Finally, the effects of a military zone between the center and an exit point (see Figure 5.1) were tested. The results are shown in Figure 5.8. The algorithm does set level flights on the route for it to avoid the restricted area.
Thus, taken one by one, all the constraints are well handled by the algorithm. In the next part,the effects of all obstacles at once with several routes to be designed are studied.



(a) The route without any obstacle.

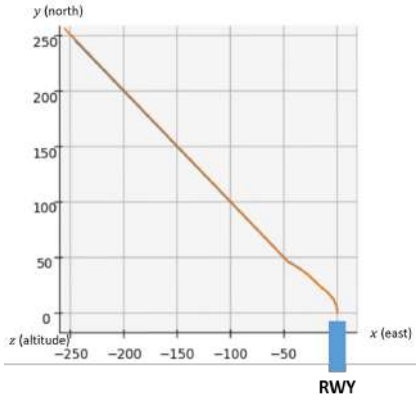


(b) The route with a mountain to avoid.

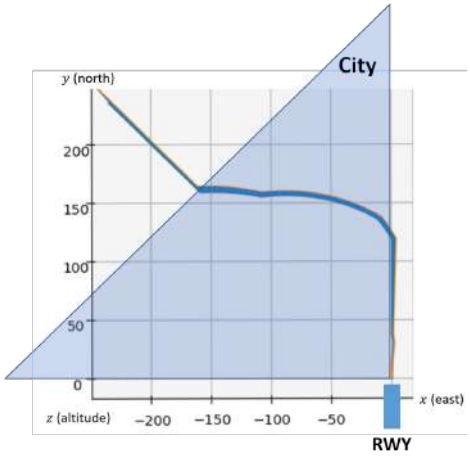


(c) The vertical profile of the route.

Figure 5.6 – The effects of a mountain on the route design.

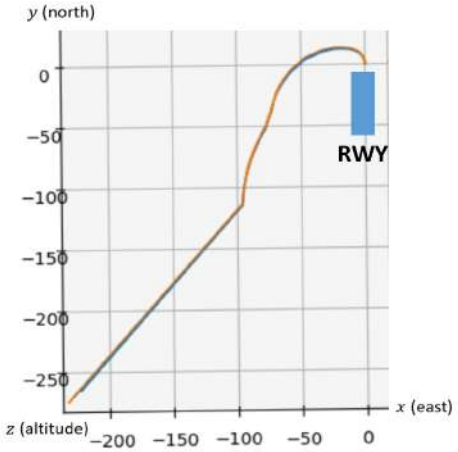


(a) The route without any obstacle.

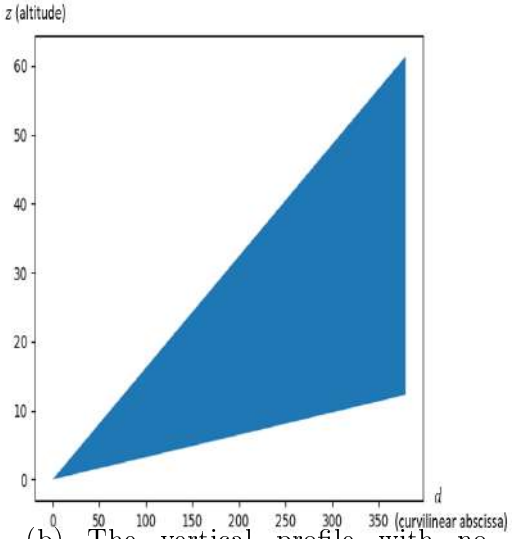


(b) The route with a city to avoid.

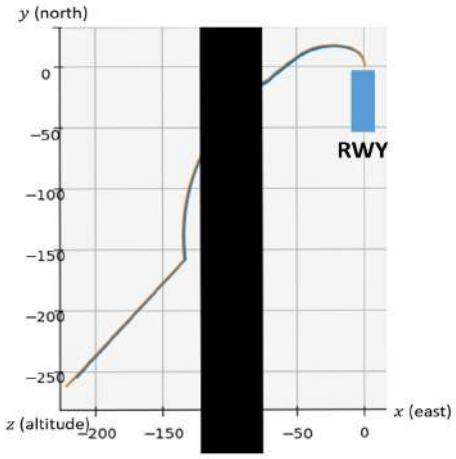
Figure 5.7 – The effects of a city on the route design. The route stays on the side of the city until its minimum altitude is high enough to fly over the city with a null cost with regard to the noise abatement criterion.



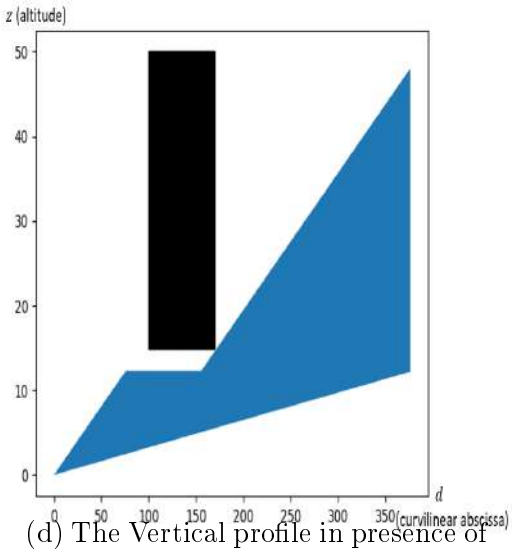
(a) The route without any obstacle.



(b) The vertical profile with no obstacle.



(c) The route with a military zone to avoid.

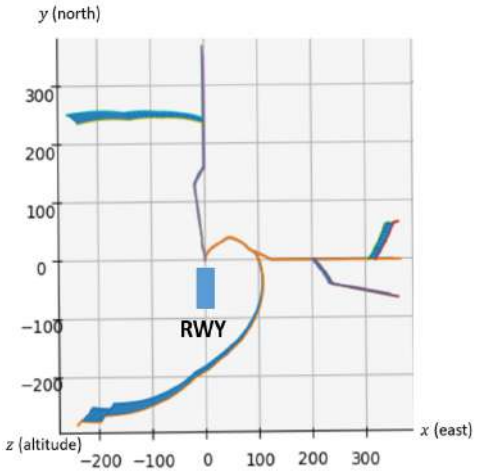


(d) The Vertical profile in presence of the zone.

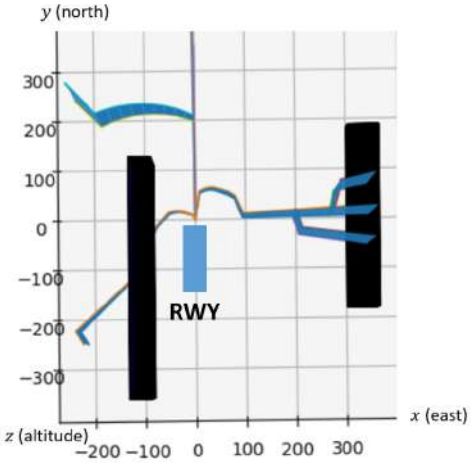
Figure 5.8 – The effects of a military zone on the route design.

5.2.2.1.2 Six routes design with all obstacles

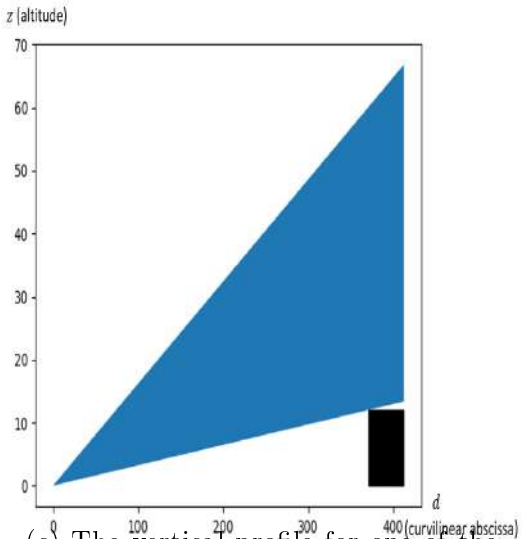
The next test that we conducted is aimed at measuring how the algorithm behaves when all the elements discussed in 5.2.2.1.1 are added. As for the other test cases, a first computation of all routes without any obstacle has been performed to set a reference. The first test uses 50 circular layers regularly spaced by 7NM, and each layer is discretized into 72 vertices (one every 5°). We also set a merge layer every 5 layers. The results are shown in Figure 5.9. It can be seen from Figure 5.9a that the routes have the



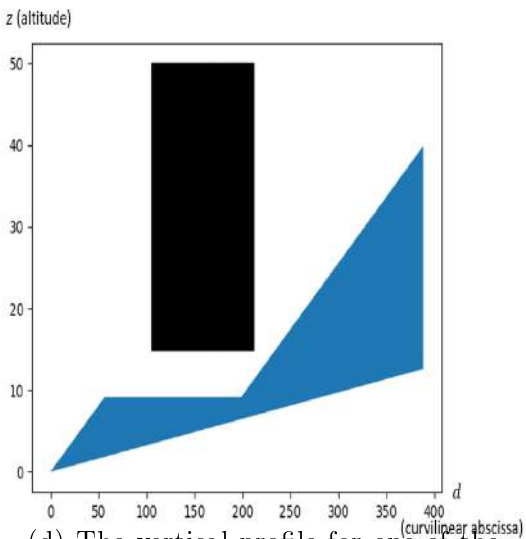
(a) The result of the algorithm without any obstacle seen from above. All constraints are met, including the limited turn constraint.



(b) The result of the algorithm with all obstacles seen from above. The routes are modified so as to meet all the constraints, such as passing over the mountain, or under the forbidden zone.



(c) The vertical profile for one of the routes above the mountain.



(d) The vertical profile for one of the routes below the military zone.

Figure 5.9 – The result of the algorithm with all obstacles at once.

expected shape: they all go in a quite straightforward way to their assigned exit points. The merge points are located near the exits when several of them

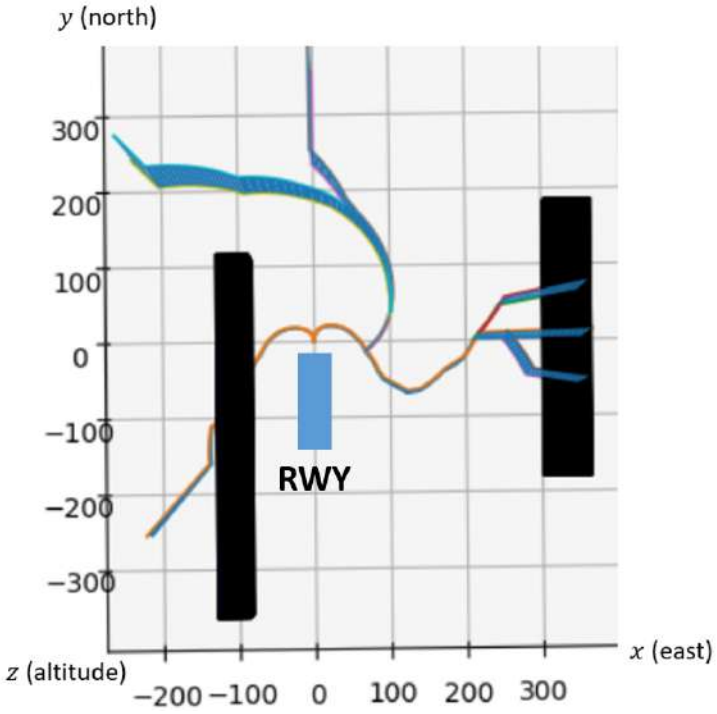
are close to each other, while the route on the left merges with the others quite close to the center, as no other exit is nearby. However, the routes are less straightforward than in the individual cases. This is due to the increase in the number of possible changes for a given iteration. In Figure 5.9b, both routes on the left have a significant change in shape. All obstacles and other routes are still avoided, the merging constraint is met, and the routes remain quite straightforward, making the solution quite relevant.
Then, the same test was run again, but with more circles (100, one every 1NM, and a merge layer every 10 layers) and more points per circle (360, one every degree), so as to measure the performance of the algorithm in terms of computation time on large instances. This also allows to see if it leads to a significant change in the shape of the routes. The obtained results are shown in Figure 5.10. All the constraints are met: the routes on the right fly over the mountain, and the route on the left doesn't pass through the military zone. However, the shape of some routes has been negatively affected compared to the previous case: both routes heading north are unnecessarily longer, and their separation at the merge point is reduced. The solution is, however, still acceptable. This test makes it explicit that a heavier discretization doesn't necessarily improve the solution. An explanation can be that in the last case, the number of possible routes is strongly increased while the algorithm runs the same number of iterations. This causes an unnecessary exploration of similar non-feasible solutions and allows for less improvements on the good ones. Table 5.4 gives some indicators on the performed tests.

We then tested the algorithm on the same instance, by changing the type of

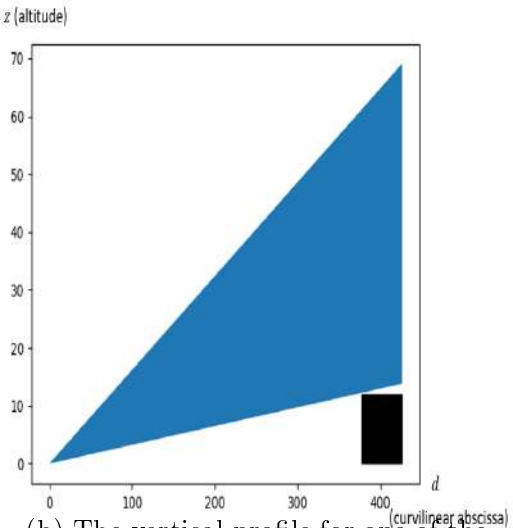
	(# of layers, # of points per layer)	Computation time	Obstacles (Table 5.1)	Routes length with obstacles / Routes length without obstacles	Graph weight with obstacles / Graph weight without obstacles
Single route	(50,72)	9 min 20s	1	417.36 / 358.61	417.36 / 358.61
	(50,72)	8 min 53s	2	394.50 / 344.79	394.50 / 344.79
	(50,72)	9 min 0s	3	376.17 / 378.83	376.17 / 378.83
Six routes	(50,72)	4 min 08s	1,2,3	2970.13 / 2701.98	1874.41 / 1673.86
	(100,360)	19 min 38s	1,2,3	2928.00 / 2568.06	2096.64 / 1601.70

Table 5.4 – The numerical results of the experiments (circular layers, one runway). The computation times are the same for both cases: with and without obstacles. The route length and graph weight criteria are worsened by 25% in average in the worst case because of the obstacles.

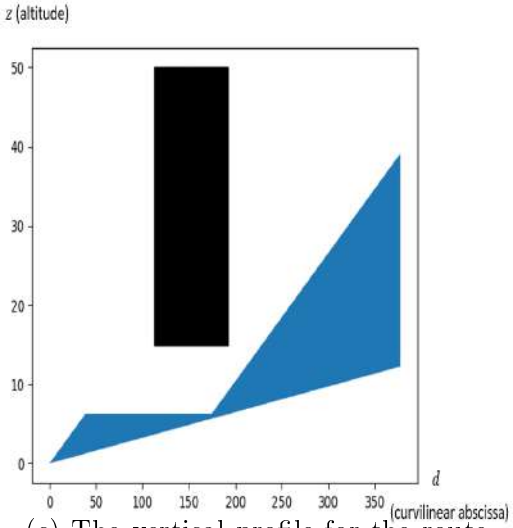
layers to squares. The aim is to see to what extent a change in the shape of the layers can affect the solutions. In this test, the squares' sides are parallel



(a) The result of the algorithm with all obstacles.



(b) The vertical profile for one of the routes above the mountain.



(c) The vertical profile for the route below the military zone.

Figure 5.10 – The result with all obstacles with a heavier discretization.

to the x and y axes. We used 86 layers progressively discretized into 48 to 360 vertices. The rest of the data in unchanged. An example of result is given by fig. 5.11 and the comparative measurements are given in Table 5.5.

The results are significantly worse than in the case of circular layers, from any point of view. The main feature that should be noticed is the lack of separation between the two routes on the right. In this case, the algorithm couldn't find any feasible solution. One of the routes is zigzagging and the city is not properly avoided.

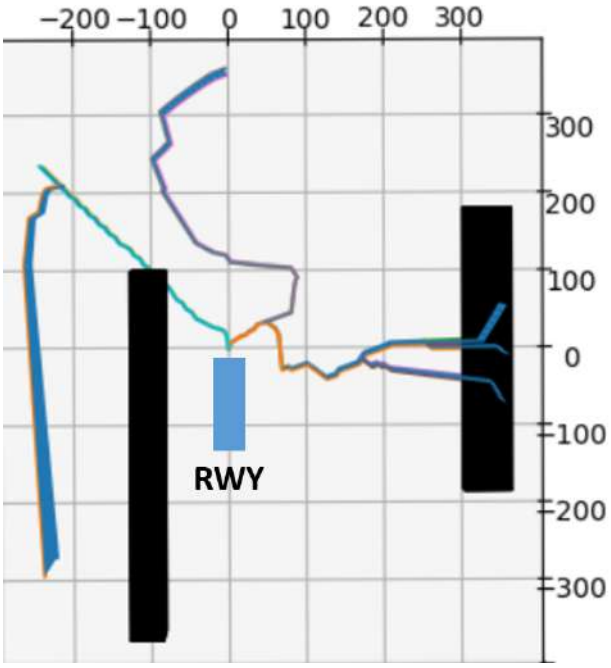


Figure 5.11 – An example of result obtained on the artificial instance with squares as layers.

Measurement	Circular layers	Square layers
Computation time	4 min 08s	29 min 27s
Route length	2970.13	3276.15
Graph weight	1874.41	2394.19

Table 5.5 – The mean results for the artificial instance with squares as layers (1 runway).

This test shows how discretization impacts the results. It is chosen arbitrarily, and to the best of our knowledge there is no way to know beforehand the type of discretization that will provide the better result. However, it has been observed in all the tests that were conducted, including the ones taken from the literature, that past a certain number of layers, the algorithm becomes less performing. Therefore, the optimal solution should be searched by changing the shape of the layers rather than their number.

5.2.2.2 The artificial instance with two runways

In order to complete the tests on the range of capabilities of the algorithm, we added a second runway to the artificial instance. The corresponding data is gathered in Table 5.6, with the features of the first runway reminded. The traffic flows are relative to their associated runway. Therefore, the total traffic flow *for each runway* is 100%. The entry points are given as polar coordinates, relatively to their respective centers. This runway is oriented by the vector (0,-1), all other parameters are unchanged. The test has been run with the two discretizations: circular and square layers. The results are presented in the next paragraphs.

5.2.2.2.1 The artificial case: two runways with circular layers

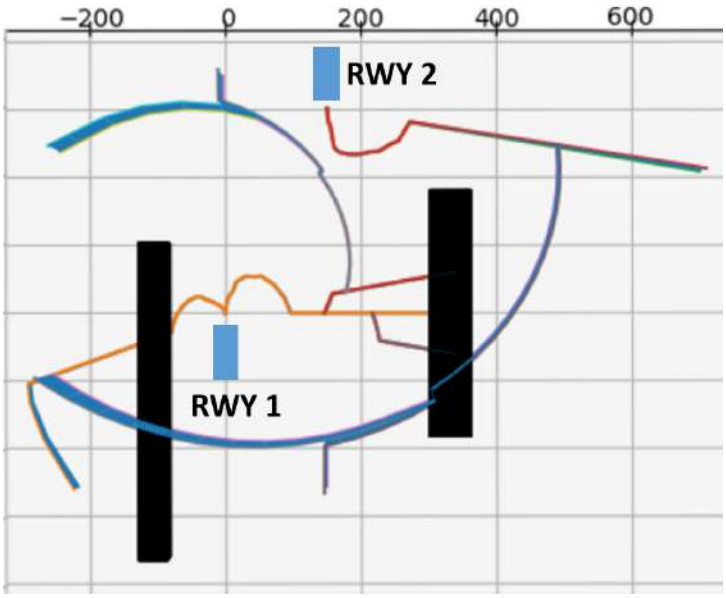
As for the configuration with only one runway, we tested a "light" and a "heavy" discretization. The parameters of these discretizations are presented in Table 5.7. We decided to keep the same discretization for the two configurations on the second runway, as the aim was to measure the performances of the algorithm under two different workloads, and the difference between the two discretizations for the first runway already providing a sufficient difference in workloads. Figure 5.12 provides examples of the results that were obtained with this configuration, and Table 5.8 the mean numerical results of the tests. The first noticeable feature is the closeness of the mean values for the route length and graph weight criteria. For the graph weight, the value is even lower for the light discretization. As for the case with one runway, it can be concluded that a heavier

Route number	Traffic flow	Graph center	Entry point
1	26.67%	(0,0,0)	(346NM, 0°)
2	24%		(346NM, 10°)
3	21.33%		(346NM ,350°)
4	13.33%		(346NM ,90°)
5	12%		(328.5NM, 135°)
6	2.67%		(328.5NM, 230°)
7	54%	(150,300,0)	(547NM, 350°)
8	27%		(547NM, 270°)
9	19%		(547NM ,225°)

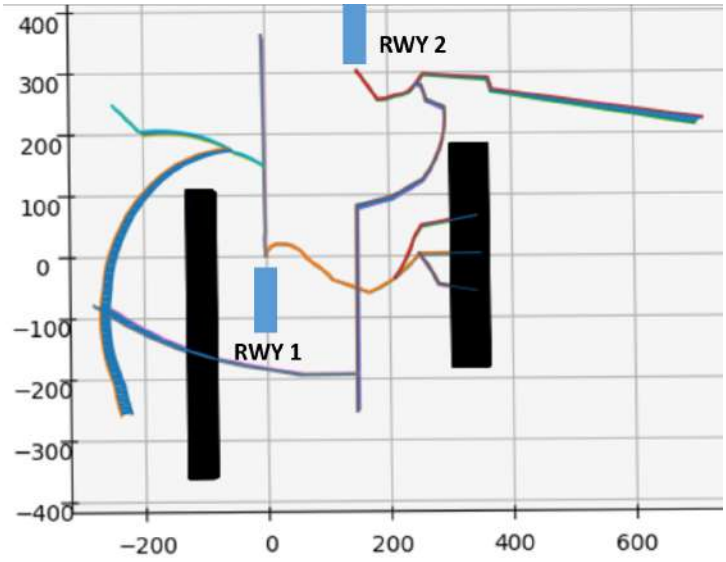
Table 5.6 – The data used for the routes in the artificial instance (2 runways).

Parameter	Light discretization	Heavy discretization
# of layers (1^{st} runway)	50	100
# of layers (2^{nd} runway)	70	70
# of points per layer (1^{st} runway)	72	360
# of points per layer (2^{nd} runway)	72	72

Table 5.7 – The data used for the artificial instance (circular layers, 2 runways).



(a) An example of result that was obtained with the light discretization.



(b) An example of result that was obtained with the heavy discretization.

Figure 5.12 – Examples of results that were obtained with layers as circles for the artificial instance (2 runways).

Measurement	Light discretization	Heavy discretization
Computation time	35 min 22s	48 min 42s
Route length	6332.40	6330.77
Graph weight	4283.12	4335.76

Table 5.8 – The mean results for the artificial instance with squares as layers (2 runways). The graph weight represents the sum of the lengths of each arc in the solution counted only once.

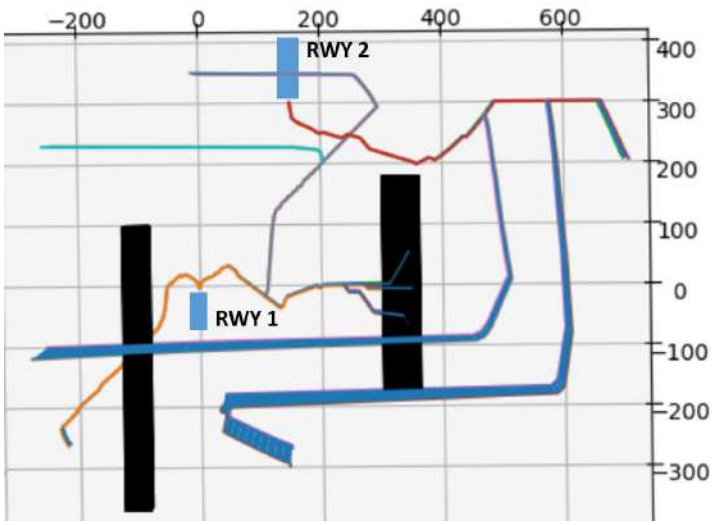
discretization does not necessarily give better results. It can also be noticed that the shape of the layers does not impact the mean time necessary to obtain an admissible solution (i.e. a solution that does not violate the obstacle constraint): a solution for the light discretization scenario is found in approximately 35 min whether the layers are circular or square, and a solution for the heavy discretization in roughly 50 min. The examples of results found by the algorithm also show that the solutions are quite similar.

5.2.2.2.2 The artificial case: two runways with square layers

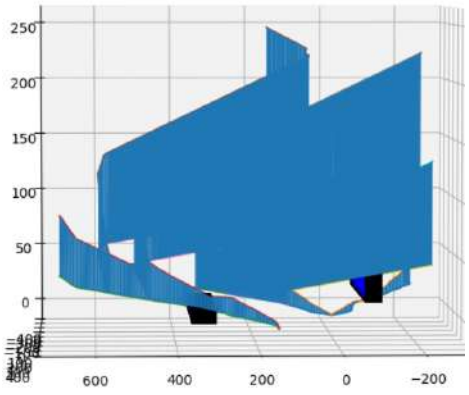
Finally, we tested the algorithm with square layers for both runways in order to compare the performances of the algorithm with the configuration with circular layers. The characteristics of the layers used are gathered in Table 5.9. An example of result that could be obtained is presented in fig. 5.13. The numerical values for the optimization criteria are gathered in Table 5.10, with the values for the approach with circular layers reminded for comparison. The results displayed in fig. 5.13 show that the algorithm becomes unable to handle correctly the considered scenario. The maximum turn angle constraint is no longer met, and the routes are significantly and unnecessarily longer, especially for the second runway. In these results, we included a "Number failed" line. It quantifies the number of solutions returned by a complete run of the algorithm that violate the obstacle avoidance constraint (in this case: when a route passes through a mountain or a military zone). The proportion of failed runs is particularly high and shows that the algorithm reaches its limits in particularly complex cases. The complexity of an instance, as the comparison between the various tests shows, is not only determined by the layout and operational constraints (obstacles, position of the entry points, cities, maximum slopes and turn

Parameter	Value
# of layers (1^{st} runway)	86
# of layers (2^{nd} runway)	110
# of points per layer (1^{st} runway)	from 48 to 360
# of points per layer (2^{nd} runway)	from 48 to 360

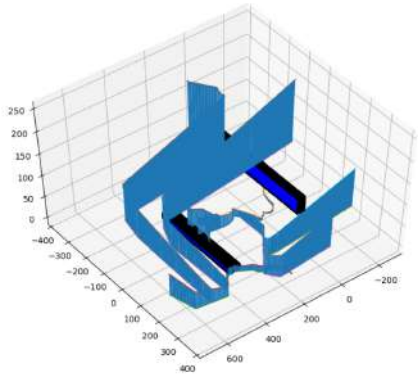
Table 5.9 – The data used for the artificial instance (square layers, 2 runways).



(a) The solution seen from above. The routes don't have a coherent shape, and one of them violates the maximum turn angle constraint.



(b) The vertical profiles of the solution. The ground obstacle and the military zone are avoided.



(c) The side view of the solution.

Figure 5.13 – A solution for the artificial case with two runways and square layers.

Measurement	Circular layers	Square layers
Computation time	35 min 22s	55 min 13s
Route length	6332.40	6976.51
Graph weight	4283.12	5470.63
Number failed	0%	75%

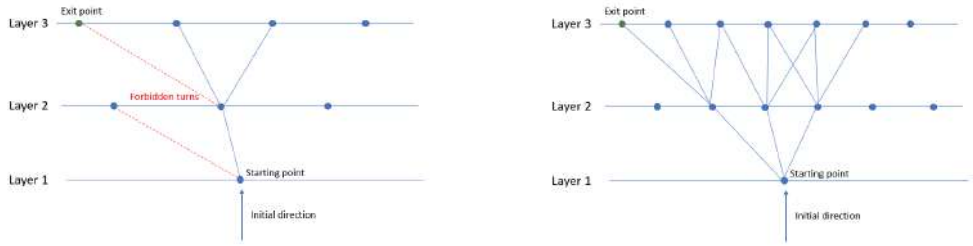
Table 5.10 – The mean results for the artificial instance with squares as layers (2 runways). The graph weight represents the sum of the lengths of each arc in the solution counted only once.

angles...), but also by the choice of the discretization to apply (shape and number of the layers, number and location of the vertices on the layers). The tests conducted on this artificial instance further demonstrated the sensitivity of the algorithm to the choice of the layers, as during our tests, two particular situations were encountered:

- The algorithm couldn't find a single admissible solution over 50 runs. The tests were stopped under the assumption that no admissible solution would be found.

- The algorithm couldn't build a given route. This means that the chosen discretization didn't allow to find a single path that led to the entry point. This case occurred more specifically when the first layers were sampled into too few vertices and thus the maximum turn angle constraint was violated during the pre-processing part of the algorithm. This case is illustrated in fig. 5.14.

In a nutshell, it can be concluded that the angular stone of the method lies in



(a) The exit point is not attainable. (b) The exit point is attainable.

Figure 5.14 – An illustration of the impact of the discretization.

the choice of the discretization: shape and number of the layers, number and points per layer. Although it has not been explicitly tested, it can also be inferred that the distribution of the points on the layers plays an important role. An idea would be to increase their number in the vicinity of obstacles and allow fewer of them in open areas.

5.3 The Stockholm instance

The first instance of a real-life scenario on which the SA based algorithm was tested is taken from the literature [114]. It involves a single runway and four STARs to be designed for the Sotckholm Arlanda airport. In [114], the routes are designed only in 2D with an integer programming (IP)-based method applied on a grid (see fig. 5.15). In their work, the authors design several solutions for the presented problem in order to find the Pareto front with regard to the route length and graph weight criteria. The solution presented in fig. 5.15 corresponds to the minimum graph weight. The drawback of this approach is its execution time: 9447 seconds (2h 37min 27s) to find the minimum weight spanning tree. The data that was used to conduct the test, with regard to the route parameters is presented in Table 5.11.

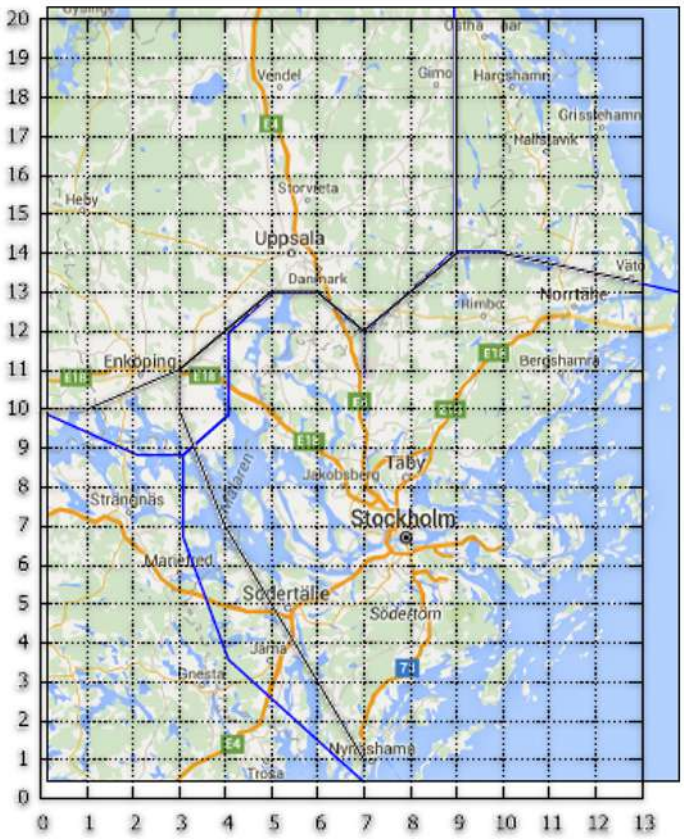


Figure 5.15 – The result from [114] and the layout used.

Route number	Traffic flow	Graph center	Entry point
1	25%	(7,11)	(14,13)
2	25%		(9,20)
3	25%		(0,10)
4	25%		(7,0)

Table 5.11 – The data used for the routes in the Stockholm instance.

Additionally, we used the following values for the parameters:

- Center = (7,12);
- The runway is oriented by the vector $(0, 1)$;
- The altitude range of the exit points is not relevant (so all ranges are accepted in the solution);
- Maximum turn angle $\theta_{max} = 45°$. It is the maximum authorized angle between two consecutive arcs in the horizontal profile;
- Minimum angle at merge points $\theta_{min} = 15°$;
- The minimum and maximum slopes α_{min} and α_{max} are not relevant;
- The number and minimum and maximum length of the level flights n^{LF}_{max}, l^{LF}_{min} and l^{LF}_{max} are not relevant;
- There is no orientation imposed on the entry/exit points.

In Table 5.12, we present the data that was used for the obstacles of the instance. Although they are cities, we represented them as "hard" obstacles, in order to comply with the representation used in [114]. In Table 5.12, each one of them is given in the form $\Big(\big((x_1, y_1); (x_2, y_2); \ldots; (x_n, y_n)\big),\ l_o,\ u_o\Big)$ where:

- $(x_1, y_1); (x_2, y_2); \ldots; (x_n, y_n)$ are the coordinates of the base polygon giving the location of the city;
- l_o is the lower altitude of the obstacle (0 being on the ground);
- u_o is the upper altitude of the obstacle.

We conducted several series of tests on this same instance: one with only two cities taken into account, and the other with all cities considered. We present and explain the results obtained in the next paragraphs.

Obstacle	(B, l, u)
Enköping	$\Big(\big((1,10);(3,10);(3,12);(1,12)\big),\ 0,\ +\infty\Big)$
Stockholm	$\Big(\big((7,6);(9,6);(9,8);(7,8)\big),\ 0,\ +\infty\Big)$
Uppsala	$\Big(\big((5,13);(6,13);(6,15);(5,15)\big),\ 0,\ +\infty\Big)$
Sodertäle	$\Big(\big((5,4);(5,5);(6,5);(6,4)\big),\ 0,\ +\infty\Big)$

Table 5.12 – The obstacles for the Stockholm instance.

5.3.1 Test case 1: only two cities taken into account

In this first test, the aim is to compare the results of our approach to [114]. Therefore, we considered only the cities of Stockholm and Enköping. We chose to use 13 concentric square layers for this test so as to match the discretization used in the literature. Square i has a $2(i-1)$ NM-long side, with square 1 being the center at (7,12) as shown in fig. 5.16. There is a vertex each 1NM on each square starting from a corner. This discretization is the same as the one used in the literature. An example of result that was obtained with our algorithm is presented in fig. 5.17, and the comparative results are presented in Table 5.13. The results presented here do not display the vertical profiles, as they were not taken into account in [114], although they were computed during the execution of the algorithm and taken into account for the obstacle avoidance and route separation. Since cities are to be completely avoided and solutions can easily be made planar, any slopes are acceptable. Both the route length and graph weight criteria are better for the solution from the literature. This is explained by three particularities. The first one is that the MILP-based algorithm is an exact approach, while the SA is not. The second reason is that in [114], the solutions are post-processed to partially become independent from the grid: all vertices that can be removed without violating the turn angle constraint are deleted (see fig. 5.18). This allows for both shorter routes and lighter graphs. Finally, our algorithm does not allow for going from a layer $\mathscr{L}_i$ to a layer $\mathscr{L}_j$ with $j \leq i$. This causes the route coming from the south in the example shown to be longer than IP route. These particularities are the reason why the solutions found with our method cannot attain the same characteristics as the ones from the literature with regard to route length and graph weight even if given more computational power.
The last feature to notice in these results is the great difference in computational time required by the solutions.

5.3.2 Test case 2: all cities taken into account

In this paragraph, we present the results obtained when including all cities of Table 5.12. The results relative to the route length and graph weight criteria are presented in Table 5.14, with a reminder of the previously presented results. As can be expected, both criteria are slightly degraded when all cities are taken into account. The shape and computing time of the average solution does not differ from the previous case with only two cities taken into account. The reason for that is that the two additional cities are already

Feature	Solution from [114]	Our solution
Route length	45.74	48.11
Graph weight	33.26	41.98
Computation time	9447s	4.54s

Table 5.13 – The comparative results for the Stockholm instance.

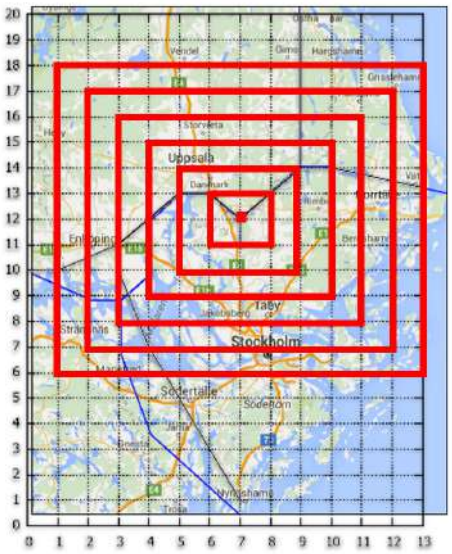
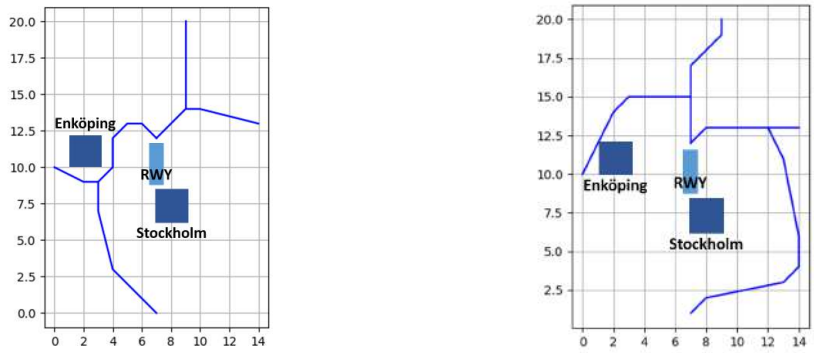


Figure 5.16 – The layers used for the first test with 7 of the 13 layers.

Feature	Solution from [114]	Our solution (2 cities)	Our solution (all cities)
Route length	45.74	48.11	52.08
Graph weight	33.26	41.98	43.56
Computation time	9447s	4.54s	3.46s

Table 5.14 – The comparative results for the Stockholm instance with squares as layers.



(a) The solution found in the literature [114].

(b) The solution found with our algorithm.

Figure 5.17 – Comparative results for the Stockholm instance.

Figure 5.18 – The post-processing technique used in [114].

avoided in the solutions from the previous test case.

5.3.3 Test case 3: circular layers

In this third test case, we decided to test another type of discretization. In this case, we tested circular layers regularly spaced by 0.5NM sampled into 360 vertices each (one every degree). There are 25 layers in total in this case. An example of result is shown in Figure 5.19. The comparative

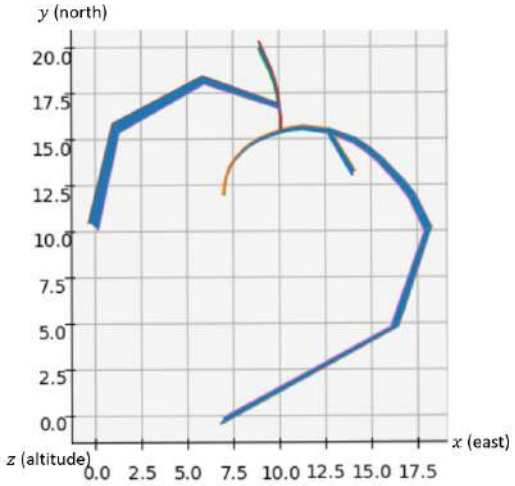


Figure 5.19 – The result of the algorithm on the Stockholm instance by using circular layers.

results are given in Table 5.15. The results are way less satisfactory than in

Feature	Our solution (square layers)	Our solution (circular layers)
Route length	52.08	73.13
Graph weight	43.56	52.4
Computation time	4.54s	125.19s

Table 5.15 – The comparative results between square and circular layers for the Stockholm instance with all cities.

the previous case, as the routes are significantly longer and occupy a wider surface. The computation time is also dramatically increased, due to the strong increase in layers and discretization points. Thus, the choice of the layout, and more particularly the shape of the layers, has a great impact on the quality of the solution given by the algorithm. The main improvement of our method over an exact approach is the reduced computation time. Although it is not critical in this particular case, since the routes are to be designed at a strategic level, which allows to take several hours, it will be an essential feature for larger instances.

5.4 The Paris Charles-de-Gaulle instance

The aim of this work is to be able to design routes for large airports, since these cases are very often more difficult to tackle than cases involving a single runway. In order to test our algorithm on a large size real-life scenario, we designed the layout of the Paris Charles-de-Gaulle airport (CDG). This instance has already been studied in [45]. Therefore, in order to establish a comparison with this approach from the literature, we kept the same configuration.
The SIDs and STARs currently in use at CDG airport are published and publicly available from Jeppesen (see the example of fig. 5.20). In our work, the initial direction for the procedures is given by the orientation of the runway it depends on. This orientation is given by the heading that an aircraft follows when using the runway. Therefore, a runway always has two possible orientations, differing by 180° from each other. This orientation is used to identify the runway, by rounding its heading to the nearest 10° and giving the first two numbers. For instance, a runway oriented towards the heading 140 (140° clockwise relatively to the magnetic north pole) will be identified as 14. Since a runway has two opposite directions, the complete identification for this example is 14/32 (see fig. 5.21). When using a runway for departure or arrivals, only one number is given, that corresponds to the heading of the aircraft. Therefore, a plane landing on runway 32 means that its heading is 320 when landing. In large airports, it is common to see two or even three parallel runways next to each other. This configuration helps absorb heavy traffic flows, as one runway can be used for take-offs and the other one for landings. When this case occurs, the runways are differentiated by letters: L for the left one, R for the right one and C for the

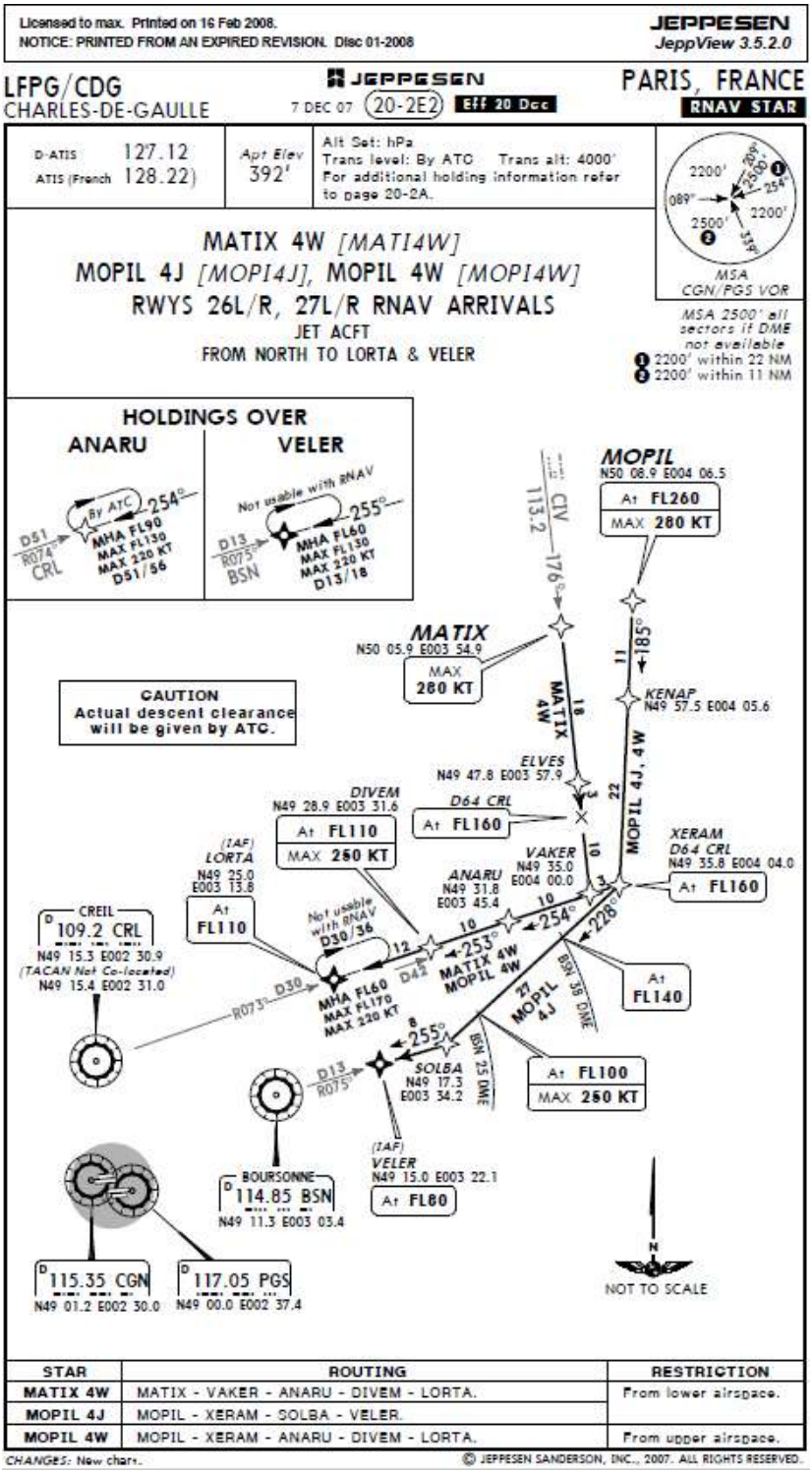


Figure 5.20 – An example of a published procedure at CDG airport.

Figure 5.21 – The identification of a runway.

center one when applicable (see fig. 5.22). In the case studied here, there are 4 runways, whose characteristics are gathered in Table 5.16 (this data is the same that is used in [45])(see fig. 5.23). Depending on the weather, the runways in use, the type of aircraft and other criteria, the operating SIDs and STARs can change. In this work, as in [45], 15 routes have been chosen. The corresponding official waypoints, as well as the estimated traffic flow, are given in Table 5.17. The data is taken from [45]. The traffic flows have been measured by counting the relative number of aircraft taking the procedures in a given time window. The graphical representation of these routes is given in fig. 5.24. The coordinates of the starting and ending points for the routes are gathered in Table 5.18.

Runway threshold	Type of procedure	Graph center	
		2D position	Height (ft)
27L	SID	(99.28, 122.95)	370
26R	SID	(100.9, 121.5)	338
27R	STAR	(111.12, 124.51)	3392
26L	STAR	(112.8, 122.66)	3316

Table 5.16 – The characteristics of the runway thresholds and starting points for the tests.

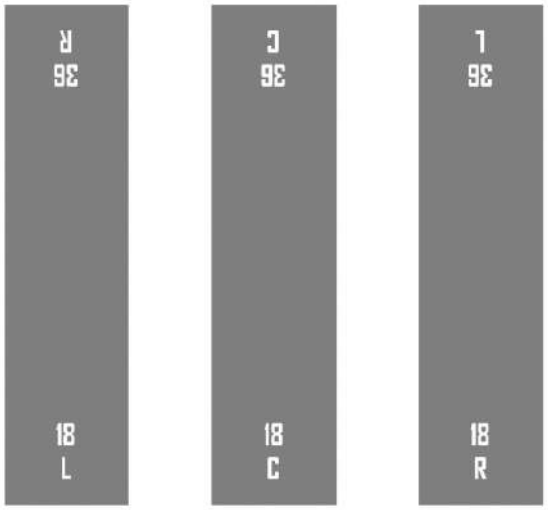


Figure 5.22 – The identification of parallel runways.

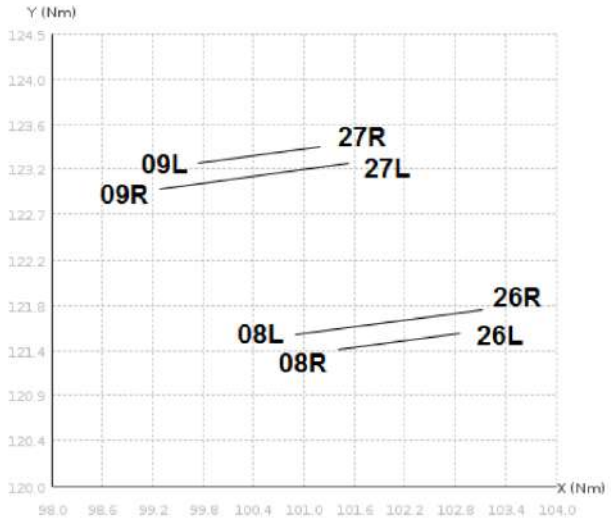


Figure 5.23 – The thresholds at CDG airport in [45].

Route	Threshold	Type of procedure	Traffic flow	Waypoints
1	27R	STAR	12.77%	DINAN, XERAM, LORTA, BSN
2	27L	SID	10.82%	PG274, PG278, NAPIX, DIKOL
3	26R	SID	10.72%	PG266, PG289, DOPAP, OKASI
4	27R	STAR	8.7%	DPE, SOKMU, KOROM, MERUE, CREIL, PG525
5	26R	SID	7.44%	PG268, RBT, ADADA, AGOPA
6	26L	STAR	7.33%	BENAR, ROMLO, BALOD, DOMUS, PG515, PG516
7	26L	STAR	7.16%	RLP, TROY, OMAKO, PG512
8	27R	STAR	6.97%	MOPIL, XERAM, LORTA, BSN
9	27L	SID	5.63%	PG276, OPALE
10	26L	STAR	5.61%	DJL, TRO, OMAKO, PG512
11	27L	SID	4.9%	PG274, PG278, LAURA, LASIV
12	27L	SID	4.76%	PG276, NURMO
13	27L	SID	3.38%	PG280, PG284, EVREUX
14	26R	SID	2.33%	PG264, PG288
15	27R	STAR	1.48%	DVL, SOKMU, KOROM, MERUE, CREIL, PG525

Table 5.17 – The routes selected for the tests for Charles-de-Gaulle airport.

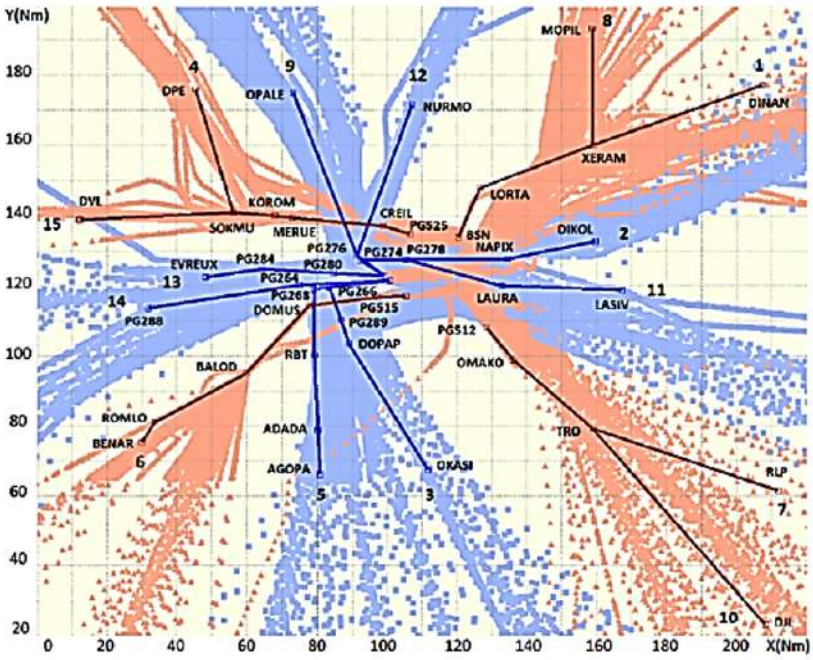


Figure 5.24 – The chosen published routes as shown in [45].

Route	Threshold	Graph center		Entry/Exit point (2D)
		2D coordinates	Height (ft)	
2	27L	(99.28, 122.95)	370	(159.87, 132.61)
9				(73.18, 175.07)
11				(167.34, 118.76)
12				(107.11, 171.69)
13				(48.26, 122.38)
3	26R	(100.9, 121.5)	338	(111.77, 67.03)
5				(80.87, 66.08)
14				(31.89, 113.55)
1	27R	(111.12, 124.51)	3392	(207.18, 177.27)
4				(45.4, 176.02)
8				(158.66, 193.36)
15				(12.31, 138.75)
6	26L	(112.8, 122.66)	3316	(29.88, 75.39)
7				(211.86, 61.57)
10				(208.23, 23.08)

Table 5.18 – The routes with their associated starting and ending points for the Charles-de-Gaulle instance.

The additional characteristics of this instance are as follows:

- The altitude range of the exit points is not relevant (so all ranges are accepted in the solution);
- Maximum turn angle $\theta_{max} = 30°$;
- Minimum angle at merge points $\theta_{min} = 15°$;
- Minimum slope (SID) α_{min}^{SID}: 7.0%;
- Maximum slope (SID) α_{max}^{SID}: 11%;
- Minimum slope (STAR) α_{min}^{STAR}: 1.6%;
- Maximum slope (STAR) α_{max}^{STAR}: 4.2%;
- The maximum number of level flights $n_{max}^{LF} = 4$ per runway threshold (so 16 in total);
- The minimum length of the level flights was set to $l_{min}^{LF} = 10NM$ and the maximum length l_{max}^{LF} is not relevant here;
- There is no orientation imposed on the entry/exit points.

Finally, we set the city of Paris as an obstacle, in order to proceed on the same base as the one that was used in [45]. It is represented as a cylinder with the following characteristics:

- The base is a circle centered on (92.43, 113.16) with a 5NM-radius;
- The lower altitude is 0;
- The higher altitude is 50,000 ft.

In the rest of this section, we provide the results that were obtained with both circular and square layers, and we establish a comparison between or results and those from the literature, in terms of route length and graph weight.

5.4.1 The Paris CDG case: a comparison with the literature

In the approach tested in [45], the problem is solved by using a Branch-and-Bound algorithm to decide for each route the way to avoid obstacles when there are any (refer to Chapter 2 for the details of this approach). This approach allows to minimize the route length criterion in a search space where all obstacles are circular, in 2D. The results obtained are shown in fig. 5.25. The main drawback of this approach is that it lacks a management of the merging points, and all routes merge on the equivalent of the *center* here, which makes it unusable in a real-life scenario. The results of our approach is shown below.

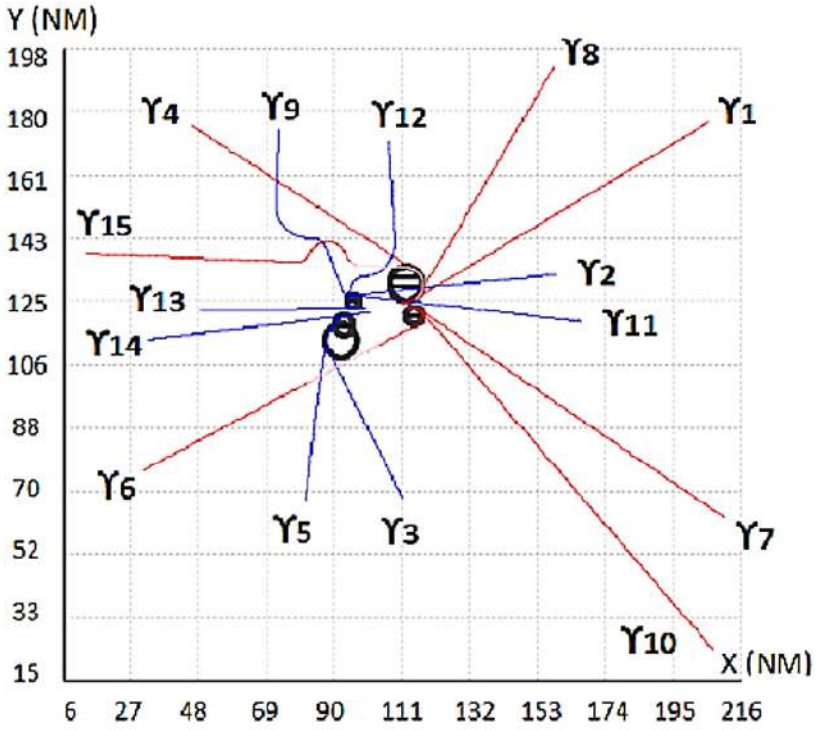


Figure 5.25 – The routes designed with the method from [45].

5.4.1.1 The CDG instance: square layers

We first tested the algorithm with square layers. These are set as shown in Table 5.19. In this case, the layers' sides are parallel to the x and y axes, and are discretized into 120 to 360 points (proportionally to their number starting at 2, so that, for instance, if there are 12 layers, $\mathscr{L}_6$ will be sampled into 216 vertices). Another point to emphasize is that whenever two entry or exit points belonging to the same graph are too close to each other, they are both put on the same layer, which is then located at mid-distance from both points. In this work, two points are considered too close to each other whenever the distance between them and the center differs by less than 1NM. The distance chosen depends on the type of layer:

- If the layers are square, the distance is the L^∞-norm, which is the maximum between the x and y values taken positively;
- For circular layers, which will be addressed later, the distance will be the usual euclidean distance.

The layers are always spaced by a constant distance. For instance, if layer $\mathscr{L}_i$ has a 2NM-long side and $\mathscr{L}_{i+2}$ has a 4NM-long side, then $\mathscr{L}_{i+1}$ has a 3NM-long side. In Table 5.19, the "Layers interval and step" column gives a starting and an ending layer, along with the distance step between two consecutive layers. With the same example as above, the column will indicate: $\mathscr{L}_i \rightarrow \mathscr{L}_{i+2} : 0.5$. The step given is the increase in the maximum x (or y) distance from the center, so half of the increase in side length (see fig. 5.26). The number given to the gates are the ones that correspond to their path (see Table 5.18). An example of result obtained with this

Threshold	Total number of layers	Layers interval and step	Gates involved	Layer of the gates
27L	17	1 → 11 : 4.874	12	11
		11 → 12 : 2.28	13	12
		12 → 13 : 1.1	9	13
		13 → 15 : 4.235	2	15
		15 → 17 : 3.735	11	17
26R	18	1 → 14 : 4.2265384	6	14
			7	14
		14 → 18 : 3.5162502	10	18
27R	21	1 → 14 : 5.055384615	4	14
		14 → 15 : 3.13	8	15
		15 → 20 : 5.442	1	20
		20 → 21 : 2.75	15	21
26L	25	1 → 21 : 4.146	3	21
		21 → 25 : 4.1	5	25
			14	25

Table 5.19 – The structure of the square layers for the CDG instance. The distance step between two consecutive layers is given, as well as the layer on which each route starts or ends.

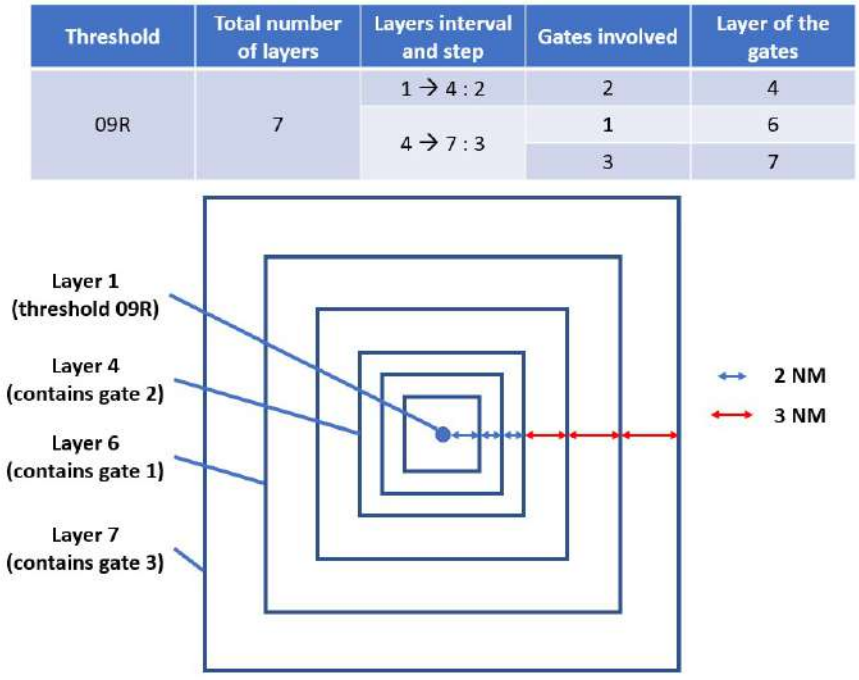


Figure 5.26 – The illustration of the functioning of Table 5.19 with a toy example. Two different spacings are used: 2 and 3NM, resp. from layer 1 to 4 and from layer 4 to 7. Gate 2 is located on layer 4, gate 1 on layer 6 and gate 3 on layer 7.

discretization is shown in fig. 5.27.
The results are quite satisfactory, since they present the following characteristics:

- The city is avoided;
- There is no conflict between the routes;
- The merging points are separated;
- The merging points are well located.

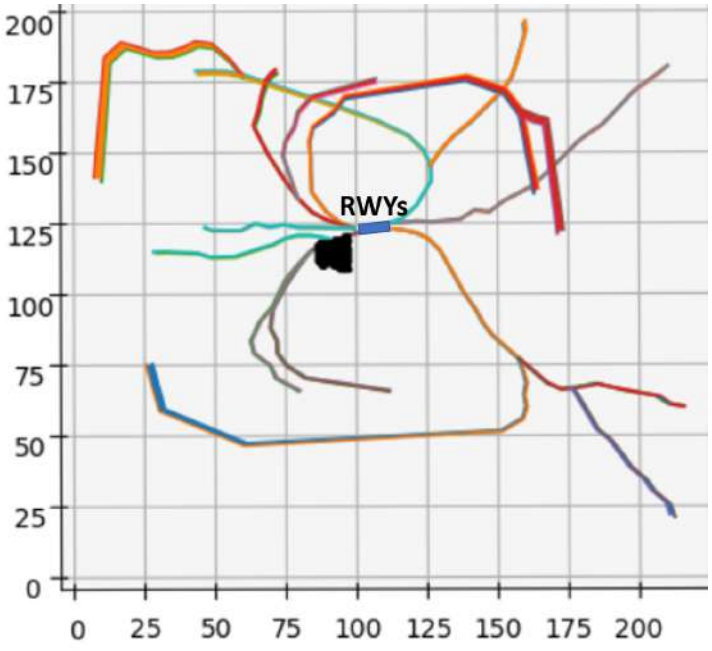
However, in this series of tests, it is to be noted that more than half of the results were rejected and the test have been performed again because the solution was not feasible. More precisely, 62% of the tests failed in that way. This means that although the city represents but a small portion of the area, the algorithm struggles to find routes that avoid it. This is mainly due to the need to design many routes, and thus, in top of the city, all conflicts between them are to be avoided, which reduces greatly the possibilities of design.
In order to test other possibilities, we then ran the same tests, with circular layers.

5.4.1.2 The CDG instance: circular layers

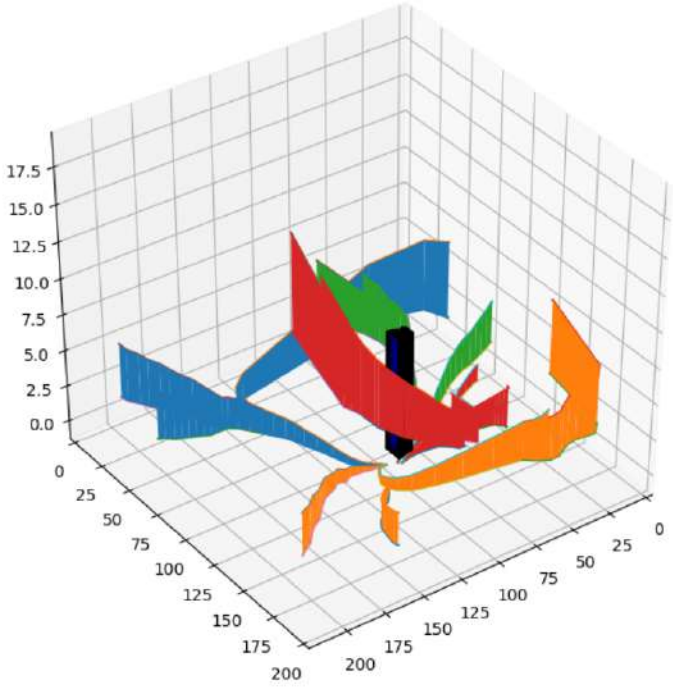
The previous test cases showed that the shape of the layers play an important role in the outcome of the algorithm. Incidentally, we decided to run the tests in the same configuration, but with circular layers. Their layout is given in Table 5.20. In this table, the mark "x" means that no gate is involved in the described layers. With all other parameters being equal to those in the square layers test case, an example of result is shown in fig. 5.28. We observe that the routes are significantly less direct, and the resulting set occupy more space than in the square layers case. It can also be noted that some of the routes adopt a zigzagging behavior towards their end. This can be due to the number of routes to design: the priority of the algorithm is to avoid the conflicts between the routes rather than designing them straight. This behavior can be smoothed with a post-processing of the result. In this series of tests, only 6% of the results were rejected due to conflicts. This is significantly lower than in the case of square layers. This difference is discussed at the end of the chapter.

5.4.2 The Paris CDG case: adding a forbidden zone

In order to test further the capacities of the algorithm, we ran it again on the same instance, with the same two sets of layers. The zone that was added has the following characteristics:

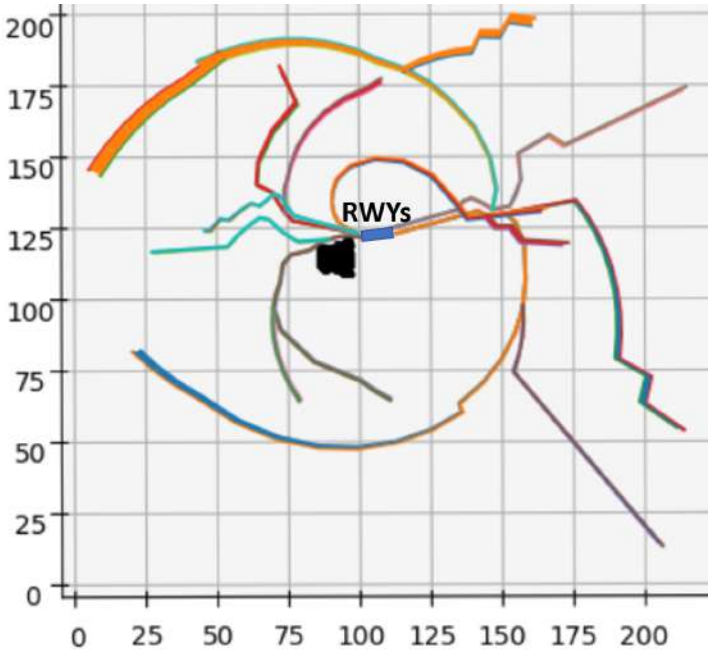


(a) 1st view.

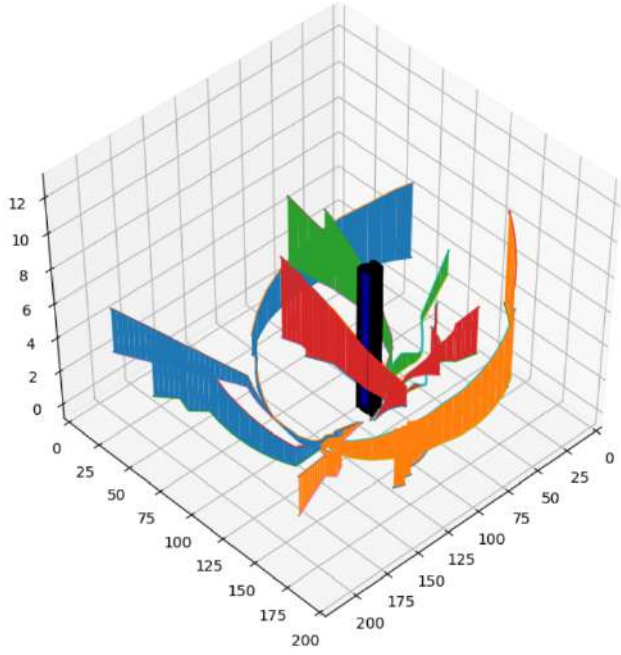


(b) 2nd view.

Figure 5.27 – The results with square layers for the CDG instance (SIDs in blue/green, STARs in red/orange).



(a) 1st view.



(b) 2nd view.

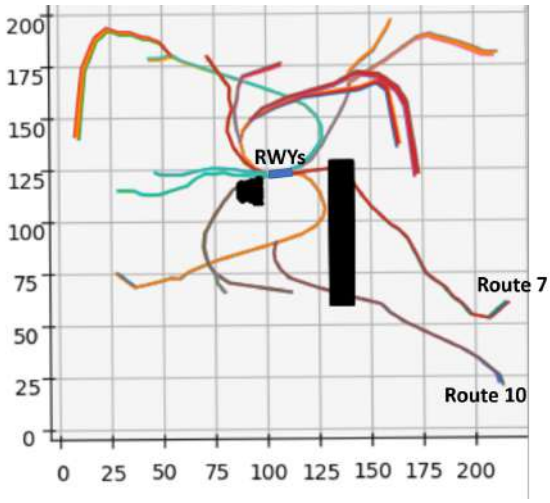
Figure 5.28 – The results with circular layers for the CDG instance (SIDs in blue/green, STARs in red/orange).

Threshold	Total number of layers	Layers interval and step	Gates involved	Layer of the gates
27L	23	1 → 2 : 4.36493188	x	x
		2 → 17 : 3	12	17
		17 → 18 : 1.65825207	13	18
		18 → 20 : 3.633323535	9	20
		20 → 21 : 3.06539451	2	21
		21 → 23 : 3.41681383	11	23
26R	21	1 → 2 : 4.7756	x	x
		2 → 18 : 3.388456	3	17
			5	18
		18 → 21 : 3.5126	14	21
27R	36	1 → 2 : 5.58456376	x	x
		2 → 28 : 3	4	28
			8	28
		28 → 33 : 3.249252328	15	33
		33 → 36 : 3.2515089	1	36
26L	46	1 → 2 : 5.44725926	x	x
		2 → 32 : 3	6	32
		32 → 39 : 2.9907397	7	39
		39 → 46 : 3.077382442	10	46

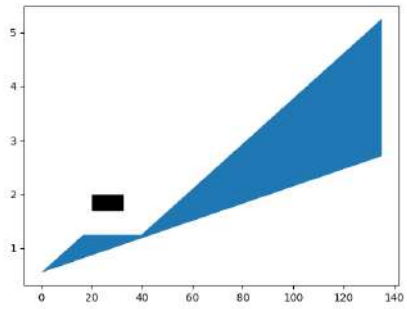
Table 5.20 – The disposition of the circular layers for the CDG instance.

- The base is a rectangle whose angles are at coordinates (130,60), (140,60), (140,127), (130,127);
- The lower altitude is 1.7NM;
- The higher altitude is 2NM.

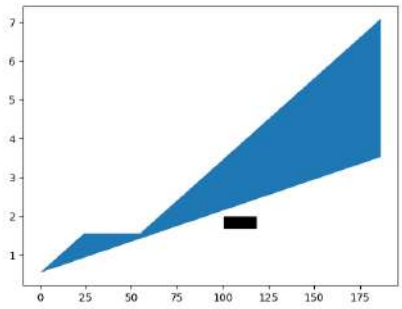
The other parameters were left unchanged. An example of result is given by fig. 5.29. In this figure, some vertical profiles are displayed, showing how the forbidden zone is avoided by the routes. In comparison with the test case without the forbidden zone, the shape of the routes is less satisfactory, especially in the positioning of the merging points. For instance, routes 7 and 10 (the STARs that avoid the zone) could have merged much closer to their respective gates, as these gates are close to each other. It would have been better to have only one route pass under the zone, that would split only afterward. This result can be explained by the need for the algorithm to avoid the forbidden zone, on top of the city and the conflicts between routes. By referring to the mathematical formulation of the problem, its priority is to find solutions that don't violate the constraints, and then comes the optimization. In this series of tests, more than 78% failed. The main cause for failure remains the non-avoidance of the city (53% of the total number of tests), the avoidance of the forbidden zone representing the other 25%.
We then tested the algorithm in the same configuration with circular layers. An example of result is given in fig. 5.30. The given vertical profiles also belong to the routes avoiding the forbidden zone. As for the square layers



(a) The routes designed for the CDG instance with a forbidden zone and square layers.

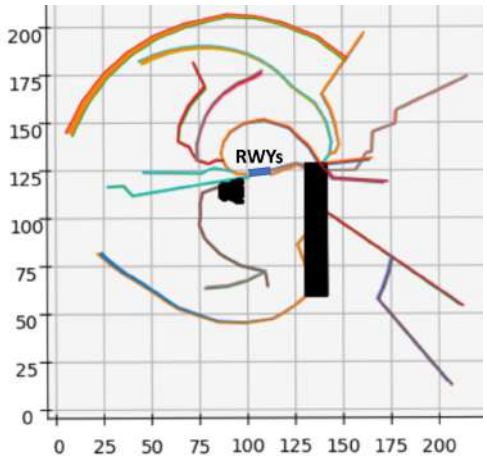


(b) The vertical profile of route 7.

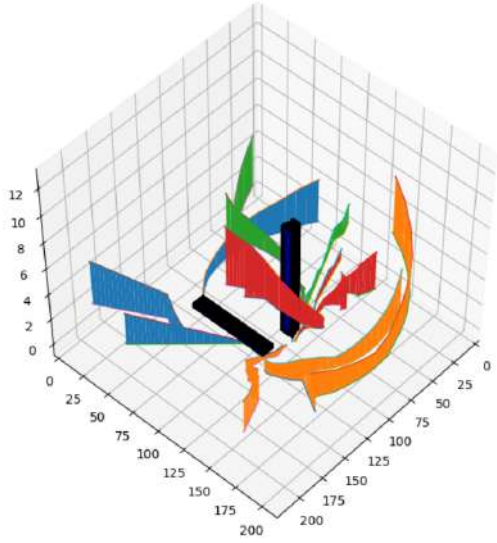


(c) The vertical profile of route 10.

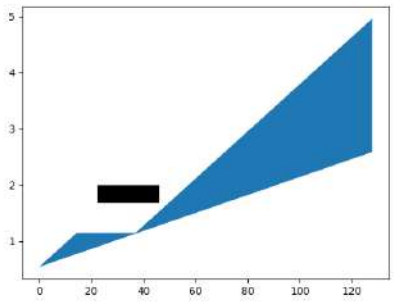
Figure 5.29 – The results with square layers for the CDG instance with a forbidden zone.



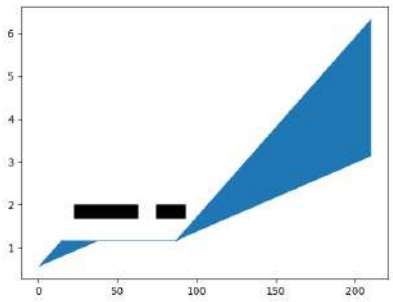
(a) The routes designed for the CDG instance with a forbidden zone and circular layers (1st view).



(b) The routes designed for the CDG instance with a forbidden zone and circular layers (2nd view) (SIDs in blue/green, STARs in red/orange).



(c) The vertical profile of route 7.



(d) The vertical profile of route 6.

Figure 5.30 – The results with circular layers for the CDG instance with a forbidden zone.

case, we observe that the results are less satisfactory in terms of shape of the routes. The explanation is the same as before. In this series of tests, only 32% of the tests failed, with 23% due to the presence of the forbidden zone and the remaining 9% to the non-avoidance of the city. Although it represents almost a third of the total number of tests run, this number is very low compared to the previous case.

5.4.3 Comparative results for the CDG instance

Over all series of tests, measurements were taken in order to compare numerically our results to the ones from the literature, and from the real-life implantation of this case. The numerical results are gathered in Table 5.21. The measurements for the standard routes are taken from [45]. A first result to consider is the difference in the route length criterion, which is higher in our work than in the published routes as well as the work from [45]. This difference with the work from the literature is explained by the fact that the latter is explicitly designed so as to minimize this criterion, by locating all merging points on the runways' thresholds, and thus it does not take into account the merging criterion. The difference with the published routes is mostly due to two elements:

- Our algorithm is not able to connect a layer i to a layer j with $j < i$;
- We set a maximum turning angle smaller than the one observed on the published procedures.

As a consequence, several routes are lengthened, mostly to avoid conflicts. This behavior can be observed for routes 4 and 15, route 6 and routes 2 and 11. In a minor way, the routes from our solution are not as straight as in the published solution. The results on the graph weight criterion are explained in a similar way. The last result to mention is the computation time required to design the solutions. Within our own results, there is a significant difference between the two choices of layers (circles or squares). This difference is due to the high proportion of failed results provided by the square discretization, resulting in almost twice the number of iterations necessary to obtain the same number of admissible solutions in that case.

Measurement	Published routes	B&B approach ([45])	SA approach ([45])
Route length	1358.26	1313.49	1295.51
Graph weight	NC	1313.49	1295.51
Computation time	NC	9.8s	20 min

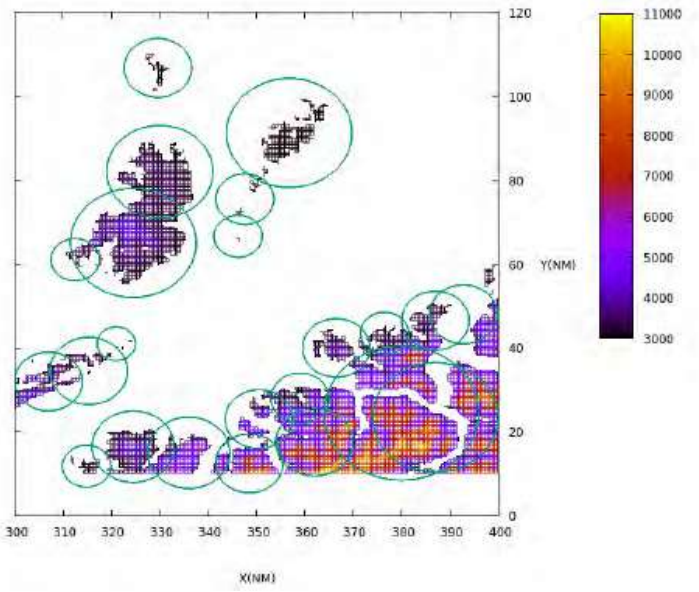
Measurement	Square layers		Circular layers	
	Without zone	With zone	Without zone	With zone
Route length	1910.38	1992.53	1972.12	1926.67
Graph weight	1508.70	1558.42	1502.52	1470.16
Computation time	13 min 45s	20 min 02s	4 min 22s	6 min 17s
Number failed	72.22%	78.57%	0%	57.75%

Table 5.21 – The mean numerical results for the Charles-de-Gaulle instance.

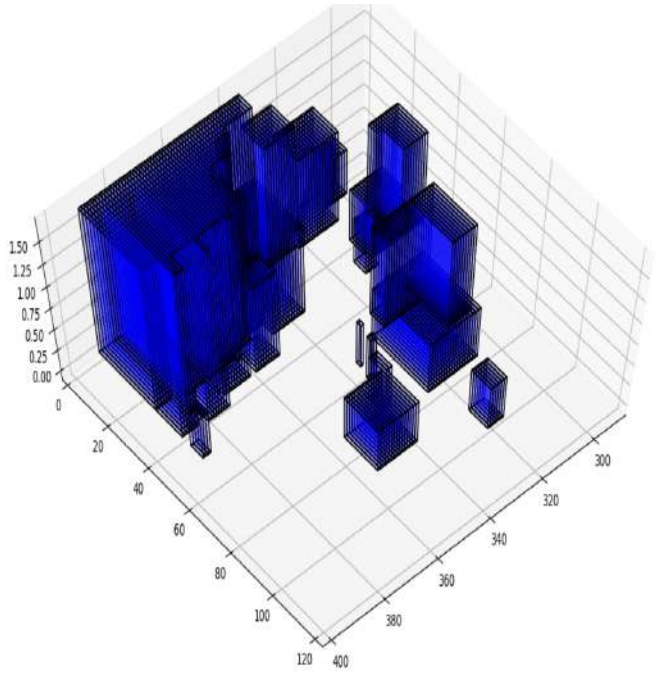
The computation time in our worst-case scenario is however comparable to the one from the literature using the same meta-heuristic (the Simulated Annealing). In this work, we consider that the acceptable time that can be used to find a satisfactory solution is of the order of 8 hours. Therefore our algorithm is compliant with this objective.
The main element that was put forward with this test case is the great sensitivity of the algorithm to the choice of the shape of the layers. In all cases, it managed to find solutions, but the number of trials necessary to achieve the same result varies strongly depending on this choice. Thus, it can be seen that the case with the circular layers allowed the algorithm to find feasible solutions more easily. It has also been shown that increasing the number of routes to be designed can affect negatively the shape of the routes in order to comply with the given constraints, and in particular the conflict avoidance. It should also be noted that several discretizations were tested beside the ones presented in this document. For many of them, the algorithm was unable to find a solution, either because it couldn't connect an entry/exit point to its runway threshold when constructing the graph, or because the discretization didn't allow it to find a feasible solution (all solutions would include conflicts). This observation strengthens our previous comment on the importance in the choice of the layers. The last point to mention is that we observed an important disparity in the observable quality of the results given by the algorithm even in the same series of tests. This shows that in complex cases the algorithm cannot be considered constant in the solution it provides, because of the numerous possibilities of design that it encounters. Therefore, it appears necessary to perform several tests for each given instance in order to obtain at least one fully coherent solution, i.e. a solution in which the shape of the routes is close to what a human could intuitively design (without steep turns or unnecessary detours or level flights, for instance). Empirically, we established that 20 successful tests were enough to attain this goal.
In the next section, a different layout is considered for measuring the performance of the algorithm in a case where the main difficulty comes from the terrain.

5.5 The Zurich instance

In the Charles-de-Gaulle instance, the main difficulty is to design a large number of routes. In this case, the environment was relatively simple, as the only obstacle was the city of Paris. Another situation that can be difficult to handle for our algorithm is when the terrain is not flat, for instance in mountainous landscapes. Such a situation has been tackled in [45] with the case of Zurich airport. In this instance, the main difficulty is to manage to design the routes so that the aircraft fly over the mountains. Figure 5.31a (taken from [45]) presents the landscape in the considered area, and fig. 5.31b the way that we modeled it. In this case, the grid we used was made up of 1NM×1NM squares. This instance features three runways: one for the SIDs (runway 10), and the STARs arrive on two different runways, 32 and



(a) The terrain elevation in the Zurich instance as shown in [45].



(b) Our model for the terrain elevation in the Zurich instance.

Figure 5.31 – The layout of the Zurich instance.

34. However, these runways share the same STARs, the separation occurs just before landing. Therefore, in the scope of this document, the algorithm works as if there were only one runway, oriented at 330°. The centers for the Zurich instance are given in Table 5.22. In [45], for this instance, no traffic flow is mentioned. Therefore, we chose to give the same importance to all routes. Additionally, one of the SIDs to design (path number 4) is supposed to start from runway 32/34, but the author of [45] chose to change its threshold to 10 on the grounds of keeping the design coherent. There are 9 routes to design in total. The traffic flows and waypoints are gathered in Table 5.23. Figure 5.32 displays the published routes, that are currently in use at this airport. The coordinates of the starting and ending points for the routes are gathered in Table 5.24. The coordinates are the same as in [45].
The additional features of this instance are as follows:

- The altitude range of the exit points is not relevant (so all ranges are accepted in the solution);
- Maximum turn angle $\theta_{max} = 30°$;
- Minimum angle at merge points $\theta_{min} = 15°$;
- Minimum slope (SID) α_{min}^{SID}: 7.0%;
- Maximum slope (SID) α_{max}^{SID}: 11%;
- Minimum slope (STAR) α_{min}^{STAR}: 2%;
- Maximum slope (STAR) α_{max}^{STAR}: 5%;
- The maximum number of level flights $n_{max}^{LF} = 4$ per runway threshold (so 8 in total);
- The minimum length of the level flights was set to $l_{min}^{LF} = 5NM$ and the maximum length l_{max}^{LF} is not relevant here;
- There is no orientation imposed on the entry/exit points.

With this data, the author from [45] presented two sets of results, one based on a Branch-and-Bound approach (fig. 5.33a), the other on a SA approach (fig. 5.33b). In both figures, the published routes are displayed in dashed lines, and the results obtained in solid lines. The green circles represent the

Runway threshold	Type of procedure	Graph center	
		2D position	Height (ft)
10	SID	(348.60, 46.53)	1400
32/34	STAR	(343.90, 50.66)	2615

Table 5.22 – The characteristics of the runway thresholds and graph centers for the Zurich tests.

Route	Threshold	Type of procedure	Traffic flow	Waypoints
1	10	SID	12%	ZH502, ZH526, ARTAG, GERSA
2	10	SID	11%	BREGO, VEBIT
3	10	SID	11%	ZH502, KOLUL, DEGES
4	10	SID	11%	SONGI
5	32/34	STAR	11%	RILAX, LAMAX, AMIKI, ZUE, TRA
6	32/34	STAR	11%	RAVED, NEGRA, MATIV, AMIKI, ZUE, TRA
7	32/34	STAR	11%	KELIP, ZH625, KLO, GIPOL, TRAD10
8	32/34	STAR	11%	BLM, GIPOL, TRAD10
9	32/34	STAR	11%	DOPIL, ERMUS, GIPOL, TRAD10

Table 5.23 – The routes selected for the tests for Zurich airport.

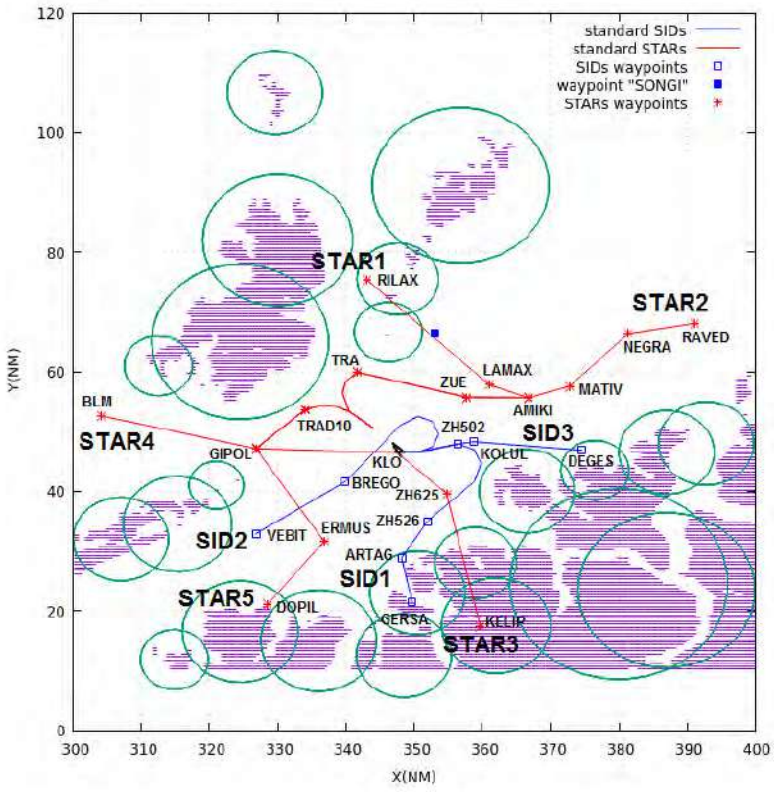


Figure 5.32 – The chosen published routes for the Zurich instance in [45].

Route	Threshold	Graph center		Entry/Exit point (2D)
		2D coordinates	Height (ft)	
1	10	(348.60, 46.53)	1400	(349.72, 21.42)
2				(326.84, 32.85)
3				(374.48, 46.86)
4				(352.95, 66.5)
5	32/34	(343.9, 50.66)	2615	(343.07, 75.39)
6				(391.14, 68.09)
7				(359.69, 17.48)
8				(304.10, 52.63)
9				(328.47, 20.99)

Table 5.24 – The routes with their associated starting and ending points for the Zurich instance.

obstacles that were chosen for the instance. As for the other cases, we tested both square and circular layers. The results obtained with our approach are presented in the next paragraphs.

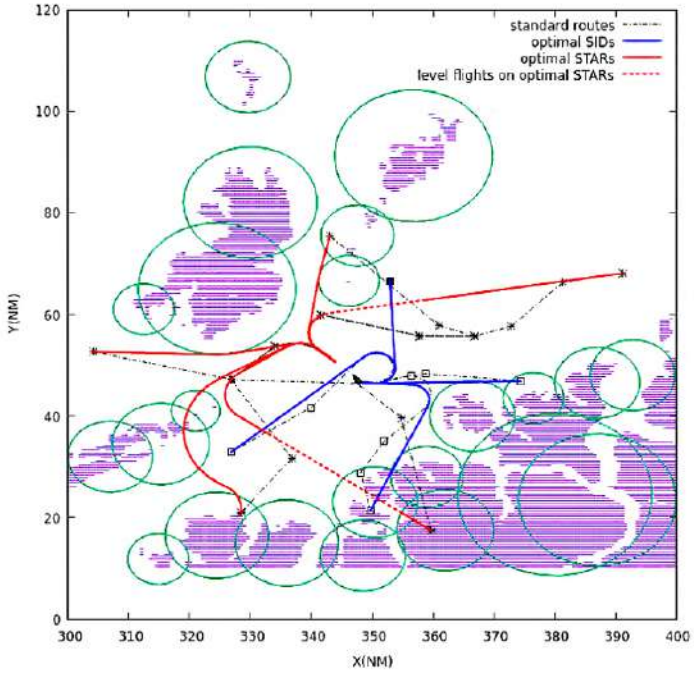
5.5.1 The Zurich instance with square layers

As the results obtained on the Charles-de-Gaulle instance were more satisfactory with the square layers, we decided to test those first. They were designed according to Table 5.25.
In this case, we tested three different discretizations of the layers:

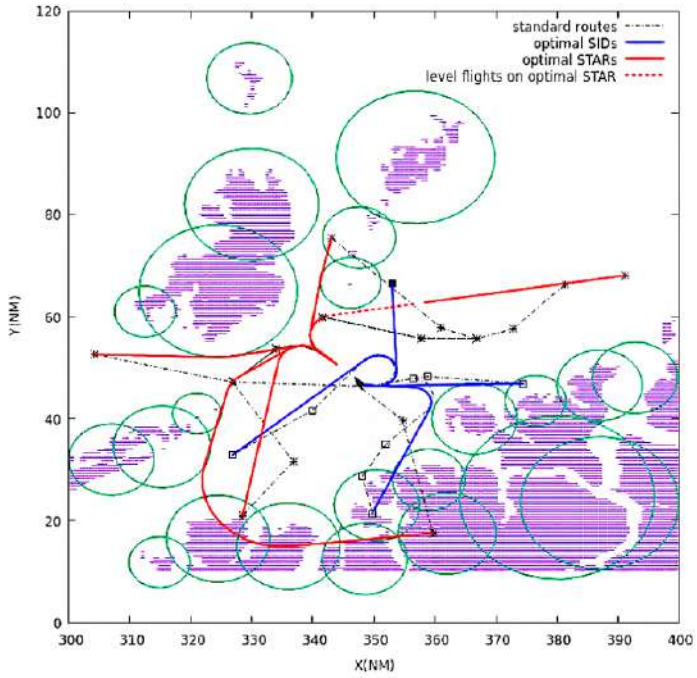
- in the first one, there are 80 to 180 vertices per layer depending on the layer's number (in the same way as for the CDG instance);
- in the second one, there are 80 to 360 vertices;
- in the third one, there are 120 to 480 vertices.

Threshold	Total number of layers	Layers interval and step	Gates involved	Layer of the gates
10	13	$1 \rightarrow 11 : 2$	4	11
		$11 \rightarrow 12 : 1.76$	2	12
		$12 \rightarrow 13 : 3.735$	1	13
			3	13
32/34	24	$1 \rightarrow 13 : 2.060833333$	5	13
		$13 \rightarrow 15 : 2.47$	9	15
		$15 \rightarrow 17 : 1.755$	7	17
		$17 \rightarrow 20 : 2.206666667$	8	20
		$20 \rightarrow 24 : 1.86$	6	24

Table 5.25 – The disposition of the square layers for the Zurich instance.



(a) The results from the literature (B&B algorithm).



(b) The results from the literature (SA algorithm).

Figure 5.33 – The results from the literature [45] on the Zurich instance.

The number and position of the layers don't change along tests, only the number of vertices per layer. This choice has been made in order for the algorithm to be able to find solutions. This aspect will be discussed at the end of this section on the Zurich instance, and more generally in the conclusion of this chapter and the discussion. The results are displayed in fig. 5.34. It appears that the results are very similar in their shapes. The mountains are avoided and no conflict is observed between the routes. The interesting comparison to consider here is the proportion of failed iterations of the algorithm (see Table 5.27). These values show that there are too few discretization points in the first case, and the algorithm is unable to easily find feasible routes, because only a small proportion of all possibilities actually leads to a feasible solution. In the third test case, the opposite problem is responsible for the number of failures: there are too many possibilities, most of which are not feasible. Therefore, the algorithm is not able to easily find a feasible solution among all possibilities. These values put forward two important aspects:

- The algorithm has reached its limits in terms of capacity to manage the terrain on this instance (a slightly more complex instance could have led to the impossibility to find any solution, at least with this type of layers);
- The choice of the density of vertices is once again a critical criterion of the performance of the algorithm.

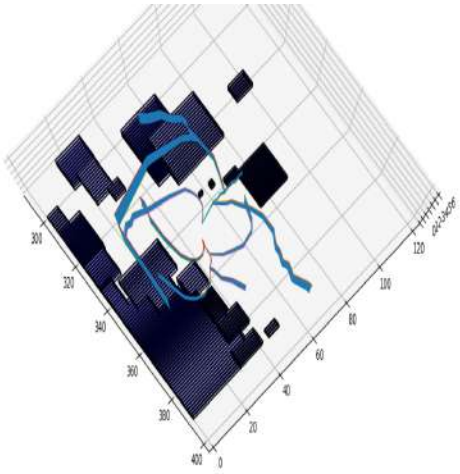
We can also mention that several other discretizations of the square layers were attempted, which couldn't provide any feasible result. This puts an emphasis on the last point.

5.5.2 The Zurich instance with circular layers

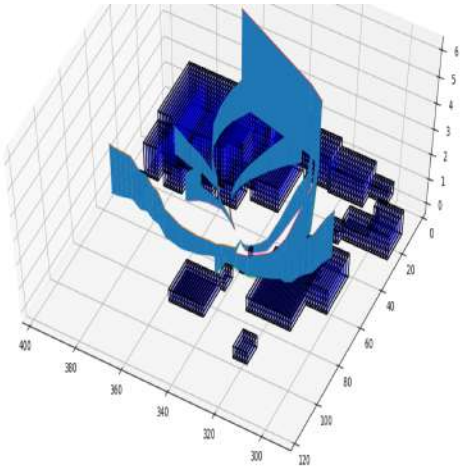
As for the other test cases, we decided to implement the layers as circles. The layers that were used are as presented in Table 5.26.

Threshold	Total number of layers	Layers interval and step	Gates involved	Layer of the gates
10	27	$1 \rightarrow 27 : 1.0$	4	21
			1	26
			2	27
			3	27
32/34	51	$1 \rightarrow 51 : 1.0$	5	26
			9	34
			7	38
			8	41
			6	51

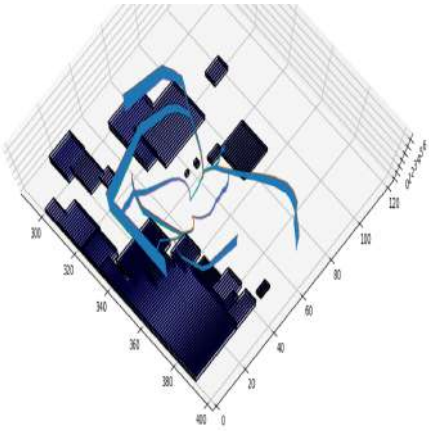
Table 5.26 – The disposition of the circular layers for the Zurich instance.



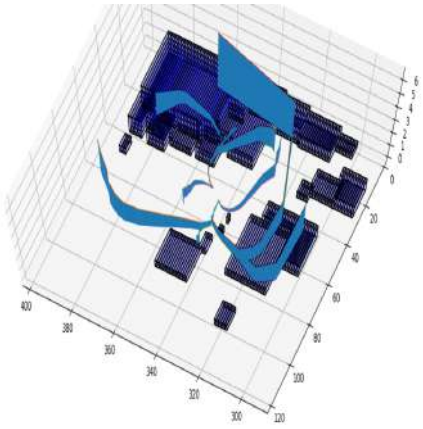
(a) The results with the first discretization (1st view).



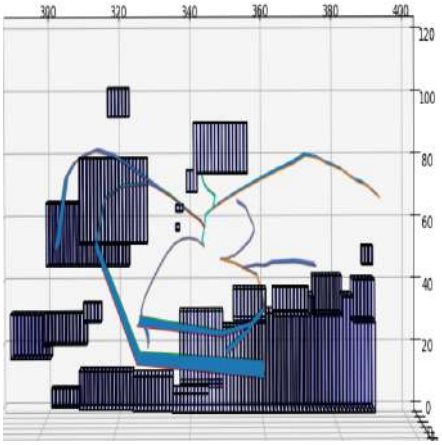
(b) The results with the first discretization (2nd view).



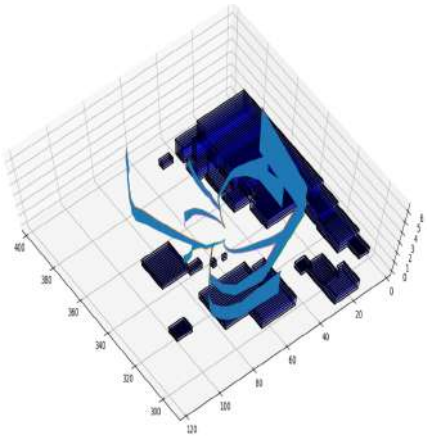
(c) The results with the second discretization (1st view).



(d) The results with the second discretization (2nd view).



(e) The results with the third discretization (1st view).



(f) The results with the third discretization (2nd view).

Figure 5.34 – The results on the Zurich instance with square layers.

As for the tests with square layers, and for the same reasons, the variation in discretization only applies to the number of vertices per layer:

- in the first case, there are 72 vertices per layer (one every 5°, see fig. 5.35);
- in the second case, there are 360 vertices per layer (one every degree).

The results obtained are presented in fig. 5.36. As for the square layers, the

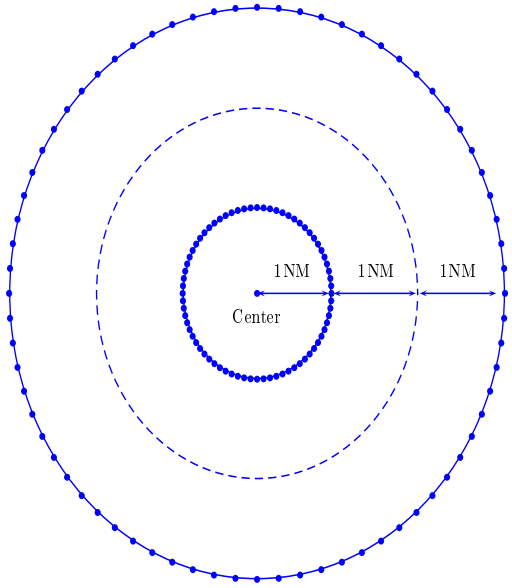
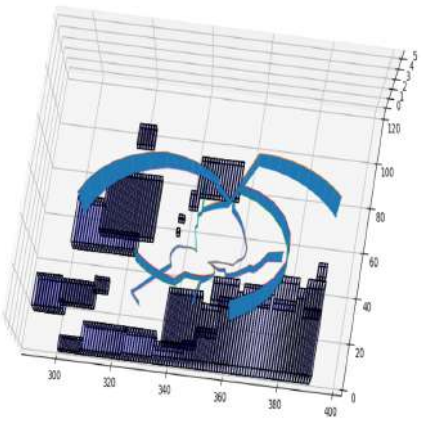
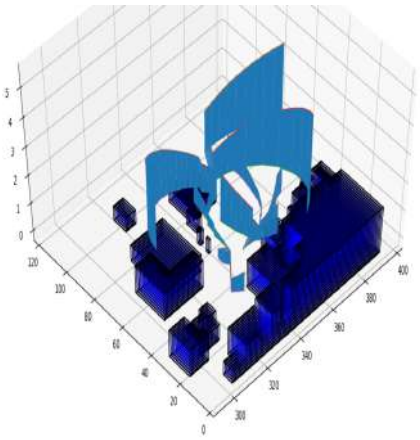


Figure 5.35 – An example of discretization for the Zurich instance in the first test case for circular layers. They are separated by 1NM each, and sampled into 72 vertices each. The same method for discretization applies for the other cases.

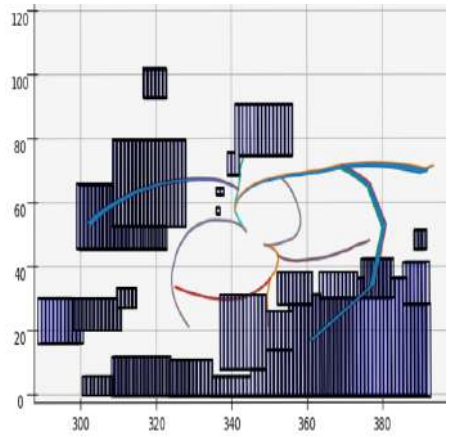
results are quite similar in shape. However, this time it can be observed that the solution from the "heavy" discretization yields a smoother result than the one with the lighter discretization, which presents irregularities. It can also be noted that in this case, the solutions provided with circular layers are visually more coherent with real-life operations than with the square layers. Again, we can look at the proportion of failed tests in these two cases in Table 5.27. These numbers, as in the Charles-de-Gaulle instance, are much lower than their counterparts in the square layers case. This tends to confirm that circular layers are more suited when the instance is complex and an admissible solution can be hard to determine. In the next paragraph, we analyze the numerical results of the tests run on the Zurich instance.



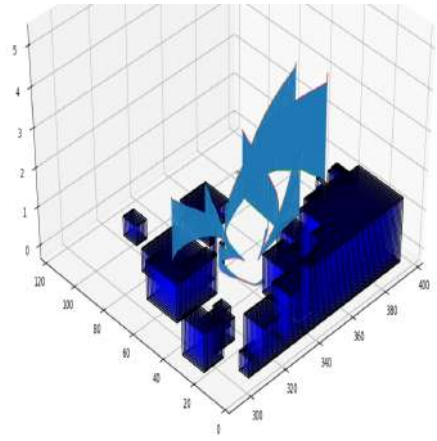
(a) The results with the first discretization (1st view).



(b) The results with the first discretization (2nd view).



(c) The results with the second discretization (1st view).



(d) The results with the second discretization (2nd view).

Figure 5.36 – The results on the Zurich instance with circular layers.

5.5.3 Comparative results and discussion on the Zurich instance

The results obtained for the tests on the Zurich instance are gathered in Table 5.27. The results show that it is possible for the algorithm to design a set of routes with a route length criterion approaching the one achieved by the procedures in use. The most remarkable feature in these results lies in the big difference between the mean computation times for the cases with circular layers. Two explanations can be brought. Firstly, it can be explained by the fact that the number of vertices per layer was multiplied by 5, which dramatically increased the time required to build the graphs as well as the arcs valuation and path search processes. The second explanation, in a minor way, is that in the first case, the algorithm ended up with a non-feasible solution in a bigger proportion than for the first case.
The last noticeable element from these results is the difference in values for the two criteria (route length, but also graph weight) between our results and those from the literature. This is mainly due to the need to manage the location of the merging points. Although one could expect the graph weight criterion to be lower in our work, this is not automatically the case, especially in instances like Zurich, where the maneuverable area is small. In this case, forcing a separation of the merging points requires to occupy more space than if it wasn't necessary. Another explanation is that for each of the two runways under consideration in this case, the entry/exit points are rather regularly spread around the centers. This implies that the routes should share little to no common parts and rather immediately take different directions, which the merging points management prevents.
The Zurich test case has shed some light on one of the most critical types of scenarios for the design of procedures with our algorithm: the complexity of the layout of the search. We mentioned above that the number and disposition of the layers weren't changed between the tests, only the number of vertices per layer. This is due to the difficulty for the algorithm to find solutions when the structure of the layers is not adapted. The following cases were encountered:

Measurement	Published routes	B&B approach ([45])	SA approach ([45])
Route length	439.52	380.44	387.69
Graph weight	NC	380.44	387.69
Computation time	NC	2.7s	12 min 50s

Measurement	Square layers			Circular layers	
	1st discr.	2nd discr.	3rd discr.	1st discr.	2nd discr.
Route length	483.95	520.30	540.28	520.94	506.68
Graph weight	413.45	436.59	464.38	392.84	407.27
Computation time	1 min 44s	1 min 35s	6 min 53s	35s	4 min 57s
Failed proportion	82%	51%	86%	20%	33%

Table 5.27 – The mean numerical results for the Zurich instance.

- If the layers are too close to one another, it is more difficult for the algorithm to design turns (as explained in the section about the artificial case). In this configuration, the algorithm was not able to connect one of the gates to its runway in the graph construction. A solution to this problem would be to increase the number of vertices per layer. However, if the number of vertices per layer increases too much, the computation time is severely worsened, and most of the solutions are very negatively affected in terms of coherence.

- If the layers are too far from one another, then there aren't enough possibilities to create paths. This often resulted in the algorithm not being able to find any admissible solution (all solutions had conflicts with the terrain).

Many configurations that were tested led to a failure to find any admissible solution. The difficulty in this test case lies in the rather short maneuverable distance between the runway thresholds and the mountains around them. This situation, as the situation of Charles-de-Gaulle airport did for the number of routes, allowed to define the limits of the algorithm in terms of terrain elevation for a real-life scenario. It also brought to our attention the effects of scattered entry or exit points around the centers, which, due to the management of the merging points, takes a toll on both the route length and graph weight criteria.

5.6 General discussion on the results

The main and most important feature of the algorithm, as put into light by the series of tests detailed above, is the choice of the layers.

The following elements were observed:

- The Stockholm instance showed the importance in the choice of the shape of the layers, even on "small" instances. The results can vary strongly depending on this choice.

- The tests on the artificial instance proved that the density of vertices also played an important role in finding a solution, even without changing the shape of the layers. It appeared that increasing the number of vertices doesn't automatically lead to better solutions, and there is no easy way to know beforehand what the best suited number of vertices is.

- The Charles-de-Gaulle instance gave an example of the limits of the algorithm in terms of capacity to produce a good solution when the number of routes to design increases strongly. There again, the importance in the choice of the shape of the layers was emphasized, the square layers giving more coherent results than the circular ones, but also leading to many more failed runs of the algorithm than their circular counterparts. It was noted that better results could be obtained by allowing wider turn angles and by being able to go back to a previous layer.

- The Zurich instance showed the limits of the algorithm in terms of terrain management, with once again more difficulties to find admissible solutions with square layers. The main difficulty for this instance was to be able to find the right number of layers and number of vertices per layer to implement. This was due to the relative smallness of the available search space compared to other large airports, and even by putting the layers close to each other, their number could be critically low for the way the algorithm was designed. In this case, the circular layers yielded better results than their square counterparts.

In all cases, the algorithm is very sensitive to the choice of the layers, both in the chosen shape and in the number of discretization points per layer. It is difficult to know in advance what type of layers will provide the most satisfactory results. However, we observed from the results of our tests that the routes have a tendency to be shaped in the form of the underlying layers: the square layers provide routes that are rather straightforward, with sharper turn angles, while the circular layers give routes that are rounder. Based on that observation, it would appear that the square layers or similar (rectangles, diamonds, trapezoids, parallelograms, regular polygon...) should be more suited for instances where the routes are expected to be straightforward (for example when there are few to no obstacles, and/or when many routes are to be designed), and that the circular layers or similar (ellipses) are more fitted for instances where it is necessary to circumvent obstacles. These assumptions are backed by the results on the tests performed for this work, but they should be investigated more thoroughly and more specifically with series of tests designed especially

to confirm or infirm them. Finally, it should be mentioned that circular layers have a better tendency to provide admissible results than their square counterparts, sometimes at the detriment of the coherence of the solution (as shown with the Charles-de-Gaulle instance), which goes in the same direction than the previous assumptions: this type of layers looks more suited for maneuverability and square ones for straightforwardness.

Chapter 6

Conclusions and perspectives

6.1 Review of the work

In this thesis, we presented a solution to automatically design Standard Instrument Departures (SIDs) and Standard Terminal Arrival Routes (STARs) at a strategic level. The method relies on the possibilities offered by the Required Navigation Performance (RNP) procedures that are being developed throughout the world to improve the way to use the airspace. The solution proposed takes many criteria into account, among which the route length, the avoidance of certain areas or the management of merging points between the routes. It was designed in order to be the closest possible to the shape of the procedures currently in use in real-life scenarios, with the minimization of the controllers' workload as a primary objective.
In our model, a route is represented by two elements. The first element is the *horizontal profile*, which consists of a sequence of geographical points in 2D (the *waypoints*) by which the aircraft are expected to fly. These points are constructed by sampling the Terminal Maneuvering Area (TMA) around the runway by the means of concentric layers, which are themselves sampled into vertices to form a graph. The horizontal profile is a sequence of points of increasing layers in that graph. The second part of the route model is the *vertical profile*, represented as a cone of altitudes associated to the horizontal profile. These altitudes encompass all the paths that aircraft following the routes could fly in the vertical plane. The vertical profile is constructed by applying a minimum and maximum slope along the horizontal profile, while taking into account possible level flights.
The obstacles are modeled as cylinders with a polygonal base, and a minimum and maximum altitudes. The cities are designed as 2D polygons associated to a population density function. They represent areas that should be avoided if possible, but that can be flown over if there is no other choice. The design of the routes is modeled as an optimization problem that we tackled with the use of an algorithm based on the Simulated Annealing (SA) meta-heuristic. For one runway, a first route is designed, and then the other routes can connect either to the runway or to a previously created route, by taking care not to connect with two routes at the same time. Each route is designed by attributing carefully chosen costs to the arcs in the graph and then by applying a shortest path search in this graph between the

desired starting and ending points. The routes are designed by decreasing order of traffic flow so as to favor the busiest ones. The same process is done for all runways, then the SA optimizes the solution. The obstacle and cities avoidance, as well as the avoidance of conflicts between routes are carried out during the evaluation process of the SA, by increasing dramatically the cost of solutions that violate one or several constraints.
This method was tested on several instances, both artificial and from real-life scenarios. These tests provided satisfactory results in the current state of Air Traffic Management, but also showed the limits of the algorithm in terms of complexity management, in terms of number of routes and terrain structure. The main element to handle is the shape of the layers used to discretize the search space. It appeared that circular layers had a better chance to give admissible results and was better tailored to instances in which avoiding obstacles is a major concern, while square layers proved to be more efficient when the main objective is to design straightforward routes, especially in an environment with few to no obstacles. In all test cases, the algorithm ran in a time span that is very acceptable in the context it is designed for. A comparison of our results with those from the literature showed that our method could be improved, in terms of route length and space occupied by the solutions. The main contribution of this thesis lies in the management of the merging points between the routes, and the variety of constraints that is taken into account, making it very close to the way actual procedures are currently designed.

6.2 Discussion and perspectives

The results showed that there is still room for improvement for the algorithm, including in terms of coherence of the routes on complex instances. Indeed, some of the routes display a zigzag behavior, are unnecessarily lengthened or connect to other routes in a non-optimal way. However, compared to the current way of designing procedures, by hand, the algorithm runs way faster. These observations make it very suited for decision-helping in the procedure design process. A first solution can be quickly provided by the algorithm, even on complex instances, and then improved by hand by experts. A first element could be to add the design of the missed approach routes to the algorithm, in order to add to the compatibility of the algorithm with the current state of air operations. Other perspectives can also be considered, and are presented in the next paragraphs.

6.2.1 Technical perspectives

Various possibilities can be considered in order to improve, or explore further the solution presented in this thesis, such as trying other meta-heuristics (genetic algorithms, ants colony algorithms, particle swarm optimization...) for instance. In this section, we present two technical perspectives that can be looked into.

6.2.1.1 Carry out an extensive study on the choice for the layers

We established that the shape and sampling of the layers used to discretize the search space play a critical role in the final solution provided by the algorithm. We were able to make assumptions on the role played by the shape of the layers, and the following ideas could help enhance the algorithm on this aspect:

- Confirm or infirm that circular or elliptic layers, and also regular polygons with a high enough number of vertices are more suited to instances in which the primary goal is to avoid obstacles;
- Confirm or infirm that square, rectangle, trapezoidal, diamond, parallelogram shapes or convex polygons that are non-regular or contain few vertices are more suited to instances where there are few to no obstacles, or where many routes are to be designed;
- Test the influence of "hybrid" shapes of layers (such as half-circles, for instance, when all entry/exit points are located in the same half plane. The half-circles can be closed by a diameter, that would pass on the center) on instances containing both situations mentioned above;
- Improve the algorithm so that it becomes able to design routes that pass several times on a given layer, allowing to make U-turns easier and thus improve the performances.

6.2.1.2 Broaden the tests to the case of a metroplex

The aim of this work is to be able to design routes for large airports. Therefore, the logical continuation is to test it on even larger instances than Charles-de-Gaulle, like the New-York TMA (see fig. 6.1). An intermediary step could be to add the airports of Paris-Orly and Paris-Le Bourget to the Charles-de-Gaulle instance. As the limits of the algorithm were seemingly close to be attained on the CDG instance, it should be expected that running the algorithm on a more complex case would yield poor results, to no result at all (impossibility for the algorithm to build one or several routes). Implementing one of the solutions mentioned in 6.2.2 would then be necessary in order to tackle larger instances.

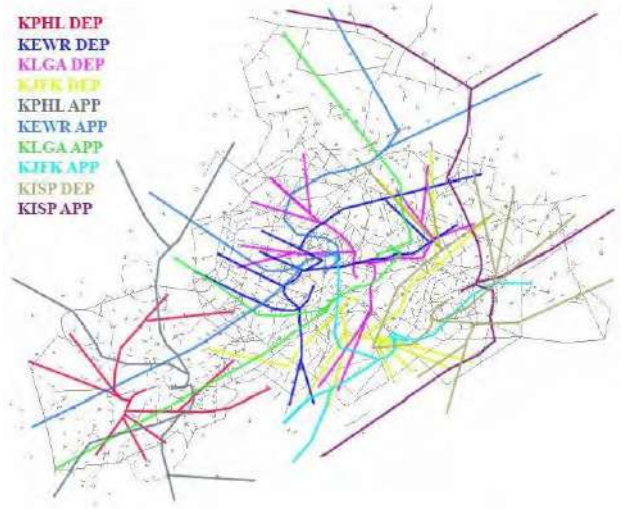


Figure 6.1 – The New-York metroplex (taken from [45]).

6.2.2 Methodological (conceptual) perspectives

6.2.2.1 Multiobjective approach

We stated in chapter 3 that the problem of SID and STAR design is multi-objective in nature. An interesting perspective could then be to modify the algorithm in a way that allows to explore the Pareto front of the solutions, rather than outputting a single solution. This would also allow for a better coverage of the various possibilities for the route design. This coverage could also help in reducing the probabilities for the final output to be an incoherent solution with regard to the current state of air traffic management.
Another way to modify the objective function could be to take the cost relative to the noise out of it, in order to be left with two criteria with the same unit (so only route length and graph weight). The part of the objective relative to the noise could then be integrated to the problem as a constraint. This constraint could be, for instance, to forbid to fly at a lower altitude than a value set for each point of the grid. This constraint would then be indistinguishable from the obstacle constraint.

6.2.2.2 New route shape modeling

In our work, we decided to design one route by relying on a graph structure with carefully chosen costs for the arcs and a shortest path search on this graph. It could be interesting to employ other means to design a route and compare them to the work presented in this thesis. Two main possibilities should be particularly interesting to consider.
The first possibility is to replace the shortest path search by a construction based on *splines*. Splines are mathematical objects that can be used to design smooth paths by interpolating them using fixed points. A spline is a sequence of polynomials of arbitrary degree, each polynomial being defined between two consecutive fixed points. A useful property of splines is that depending on their degree, regularity conditions can be imposed on

the connection points, which allows to create smooth paths. Using splines instead of arcs should take the solutions provided by the algorithm farther from current designs and closer to a mathematical modeling of paths than the ones presented in this work, but should help in implementing more efficiently the limited turn constraint.
The second possibility to design the routes in another way is to use the optimal control theory. This method takes in entry the state equations, describing the movement of an aircraft, functions describing the constraints, and an optimization function. Several methods can be used to solve the associated optimization problem, and the solution provided gives the optimal way to fly the aircraft under the optimization function provided (for example the route shortness, or the fuel consumption...). Although all routes could be designed with this method, it could also be interesting to apply it to each route, inside a Simulated Annealing-based algorithm. This would allow to modify the optimization function during the process, or for each route, allowing to explore the search space more thoroughly.

Publications

This thesis lead to two publications

- In AIAA's journal *Aerospace*: *Departure and Arrival Routes Optimization Near Large Airports*, 6(7), 80, published on July, 12$^{\text{th}}$, 2019; https://doi.org/10.3390/aerospace6070080.
- For the EIWAC2019 conference in Tokyo, Japan: *SID and STAR routes optimization near large airports*, published in HAL archive: https://hal-enac.archives-ouvertes.fr/hal-02352437, February, 8$^{\text{th}}$, 2020, HAL id: hal-02352437, version 1.

Bibliography

[1] "Global facts sheet." https://aviationbenefits.org/media/166713/abbb18_factsheet_global.pdf, 2018.

[2] "European aviation in 2040 - challenges of growth - annex 4: Network congestion." https://www.eurocontrol.int/sites/default/files/2019-05/challenges-of-growth-2018-network-congestion-21122018.pdf, 2018.

[3] "European aviation in 2040 - challenges of growth." https://www.eurocontrol.int/sites/default/files/content/documents/official-documents/reports/challenges-of-growth-2018.pdf, 2018.

[4] "Commercial market outlook 2019 - 2038." https://www.boeing.com/resources/boeingdotcom/commercial/market/commercial-market-outlook/assets/downloads/cmo-sept-2019-report-final.pdf, 2018.

[5] S. J. Undertaking, "European atm master plan - executive view," tech. rep., 2020.

[6] F. A. A. (FAA), "Nextgen implementation plan," tech. rep., 2018.

[7] E. Commission, "Air traffic management - freeing europe's airspace," tech. rep., Commission of the European Communities, 1996.

[8] C. Letondal, C. Hurter, R. Lesbordes, J.-L. Vinot, and S. Conversy, "Flights in my hands: Coherence concerns in designing strip'tic, a tangible space for air traffic controllers," tech. rep., 04 2013.

[9] S. Rathinam, J. Montoya, and Y. Jung, "An optimization model for reducing aircraft taxi times at the dallas fort worth international airport," *MDPI*, vol. 1, pp. 3470–3483, 2008.

[10] M. Zhang, Q. Huang, S. Liu, and H. Li, "Multi-objective optimization of aircraft taxiing on the airport surface with consideration to taxiing conflicts and the airport environment," *MDPI - Sustainability*, vol. 11, p. 6728, 2019.

[11] T. Degaspare, P. Paglione, and C. Marinho, "Development of a simulation environment for ground control laws design," 12 2015.

[12] S. Chaimatanan, D. Delahaye, and M. Mongeau, "A hybrid metaheuristic optimization algorithm for strategic planning of 4d aircraft trajectories at the continental scale," *IEEE Computational Intelligence Magazine*, vol. 9, pp. 46–61, 2014.

[13] O. Rodionova, M. Sbihi, D. Delahaye, and M. Mongeau, "North atlantic aircraft trajectory optimization," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, pp. 2202–2212, 2014.

[14] C. Allignol, N. Barnier, and A. Gondran, "Optimized vertical separation in europe," in *31st IEEE/AIAA Digital Avionics Systems Conference (DASC2012)*, pp. 4B3–1–4B3–10, 2012.

[15] N. Durand and J.-B. Gotteland, "Genetic algorithms applied to air traffic management," *Metaheuristics for Hard Optimization Methods and Case Studies*, pp. 277–306, 2006.

[16] S. Cafieri and N. Durand, "Aircraft deconfliction with speed regulation: New models from mixed-integer optimization," *Journal of Global Optimization*, vol. 58, no. 4, pp. 613–629, 2014.

[17] N. Barnier and C. Allignol, "4d-trajectory deconfliction through departure time adjustment," in *8th USA/Europe Air Traffic Management Research and Development Seminar (ATM 2009)*, 2009.

[18] ICAO, *PANS OPS vol II - Construction of Visual and Instrument Flight Procedures*, 2006.

[19] ICAO, *Performance-based Navigation (PBN) Manual*, 2008.

[20] ICAO, *Required Navigation Performance Authorization Required (RNP AR) Procedure Design Manual*, 2009.

[21] "Compatibility in clearances issued," in *International Federation of Air Traffic Controllers' Associations (IFACTA) 58th annual conference*, 2019.

[22] Eurocontrol, *Guidance Material for the Design of Terminal Procedures for DME/DME and GNSS Area Navigation*, 1999.

[23] Eurocontrol, *Point Merge Integration of Arrival Flows Enabling Extensive RNAV Application and Continuous Descent - Operational Services and Environment Definition*, 2010.

[24] "Point merge." https://www.eurocontrol.int/concept/point-merge.

[25] B. Delaunay, "Sur la sphère vide," *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, vol. 7, pp. 793–800, 1934.

[26] P. Maur, "Delaunay triangulation in 3d," 2002. State of the Art and Concept of Doctoral Thesis - University of West Bohemia in Pilsen, Czech Republic.

[27] G. Leach, "Improving worst-case optimal delaunay triangulation algorithms," in *4th Canadian Conference on Computational Geometry*, 1992.

[28] M. Brevilliers, "Triangulations de steiner," 2006. LMIA Seminary.

[29] Y. Thoma, *Tissu Numérique Cellulaire a Routage et Configuration Dynamiques*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 2005.

[30] D. Watel, *Approximation de l'Arborescence de Steiner*. PhD thesis, Université de Versailles Saint-Quentin-en-Yvelines, 2014.

[31] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Communications of the ACM*, vol. 22, no. 10, pp. 560–570, 1979.

[32] H. Rohnert, "Shortest paths in the plane with convex polygonal obstacles," *Information Processing Letters*, vol. 23, no. 2, pp. 71–76, 1986.

[33] J. Storer and J. Reif, "Shortest paths in the plane with polygonal obstacles," *Journal of the ACM*, vol. 41, pp. 982–1012, 1994.

[34] M. N. Bygi and M. Ghodsi, "3d visibility graph," *Computational Science and its Applications*, 2007.

[35] M. Sharir and A. Schorr, "On shortest paths in polyhedral spaces," pp. 144–153, 1984.

[36] K. Jiang, L. Seneviratne, and S. Earles, "Finding the 3d shortest path with visibility graph and minimum potential energy," vol. 1, pp. 679–684, 1993.

[37] Y.-H. Liu and S. Arimoto, "Path planning using a tangent graph for mobile robots among polygonal and curved obstacles," *International Journal of Robotic Research*, vol. 11, no. 4, pp. 376–382, 1992.

[38] M. Pocchiola and G. Vegter, "Minimal tangent visibility graphs," *Computational Geometry*, vol. 6, no. 5, pp. 303–314, 1996.

[39] F. Aurenhammer, "Voronoi diagrams - a survey of a fundamental geometric data structure," *ACM Computing Surveys*, vol. 23, pp. 345–405, 1991.

[40] "Quadtree." https://github.com/domoench/Quadtree.

[41] H. Choset, *Sensor Based Motion Planning: The Hierarchical Generalized Voronoi Graph*. PhD thesis, California Institute of Technology, 1996.

[42] B. Chazelle and L. Guibas, "Visibility and intersection problems in plane geometry," *Discrete & Computational Geometry*, vol. 4, no. 6, pp. 551–581, 1989.

[43] N. Sleumer and N. Tschichold-Gürman, "Exact cell decomposition of arrangements used for path planning in robotics," tech. rep., Institute of Theoretical Computer Science Zurich, 1999.

[44] F. Lingelbach, "Path planning using probabilistic cell decomposition," in *Proceedings of 2004 IEEE International Conference on Robotics and Automation*, vol. 1, pp. 467–472, 2004.

[45] J. Zhou, *Optimal Design of SIDs/STARs in Terminal Maneuvering Area*. PhD thesis, Université Toulouse 3 Paul Sabatier, 2017.

[46] D. Demyen, "Efficient triangulation-based pathfinding," Master's thesis, University of Alberta, 2007.

[47] A. Ait El Cadi, *Planification de Trajectoires pour une Flotte d'UAVs*. PhD thesis, Université de Montreal, 2010.

[48] P. Gallina and A. Gasparetto, "A technique to analytically formulate and to solve the 2-dimensional constrained trajectory planning problem for a mobile robot," *Journal of Intelligent and Robotic Systems*, vol. 27, no. 3, pp. 237–262, 2000.

[49] D. Delahaye, S. Puechmorel, P. Tsiotras, and E. Feron, "Mathematical models for aircraft trajectory design: A survey," *Air Traffic Management and Systems: Selected Papers of the 3rd ENRI International Workshop on ATM/CNS (EIWAC2013)*, pp. 205–247, 2013.

[50] H. Jeffreys and B. Jeffreys, "Methods of mathematical physics." Cambridge University Press, 1988.

[51] G. Farin and D. Hansford, *The Essentials of CAGD*. Natick, MA, USA: A. K. Peters, Ltd., 2000.

[52] W. Mula, "B-spline with control points/control polygon, and marked component curves." https://fr.wikipedia.org/wiki/B-spline#/media/Fichier:B-spline_curve.svg.

[53] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[54] A. Shimbel, "Structure in communication nets," in *Proceedings of the Symposium on Information Networks*, pp. 199–203, Polytechnic Press of the Polytechnic Institute of Brooklyn, 1955.

[55] L. Ford, *Network Flow Theory*. RAND Corporation, 1956.

[56] R. Bellman, "On a routing problem," *Quaterly of Applied Mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[57] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[58] D. Ferguson, M. Likhachev, and A. Stentz, "A guide to heuristic based path planning," *Proceedings of the Workshop on Planning under Uncertainty for Autonomous Systems at The International Conference on Automated Planning and Scheduling (ICAPS)*, 2005.

[59] D. M. Pfeil, *Optimization of Airport Terminal-Area Air Traffic Operations under Uncertain Weather Conditions.* PhD thesis, Massachusetts Inst. of Technology, Cambridge, MA, 2011.

[60] J. Chen, A. Yousefi, S. Krishna, B. Sliney, and P. Smith, "Weather avoidance optimal routing for extended terminal airspace in support of dynamic airspace configuration," in *31st IEEE/AIAA Digital Avionics Systems Conference (DASC2012)*, pp. 3A1–1–3A1–16, 2012.

[61] J. Chen, A. Yousefi, S. Krishna, D. Wesely, B. Sliney, and P. Smith, "Integrated arrival and departure weather avoidance routing within extended terminal airspace," in *32nd IEEE/AIAA Digital Avionics Systems Conference (DASC2013)*, pp. 1A4–1–1A4–17, 2013.

[62] M. Shakour, "Conception et implementation d'un algorithme de planification de chemin dans un jeu vidéo comportant un environnement triangularisé," Master's thesis, Université du Québec à Montréal, 2012.

[63] L. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics, vol. 79, no 3*, pp. 497–516, 1957.

[64] T. Li, J. Jiang, Z. Zhen, and C. Gao, "Mission planning for multiple uavs based on ant colony optimization and improved dubins path," in *IEEE Chinese Guidance, Navigation and Control Conference*, 2016.

[65] R. Garroppo, S. Giordano, and L. Tavanti, "A survey on multi-constrained optimal path computation: Exact and approximate algorithms," *Elsevier*, 2010.

[66] C. H. and S. M. LaValle, "Time-optimal paths for a dubins airplane," in *46th IEEE Conference on Decision and Control*, pp. 2379–2384, 2007.

[67] I. Lugo-Cárdenas, G. Flores, S. Salazar, and R. Lozano, "Dubins path generation for a fixed wing uav," in *International Conference on Unmanned Aircraft Systems (ICUAS 2014)*, pp. 339–346, 2014.

[68] A. Richards, "Trajectory optimization using mixed-integer linear programming," Master's thesis, Massachusetts Institute of Technology, 2002.

[69] J. Bellingham, M. Tillerson, A. Richards, and J. How, *Multi-Task Allocation and Path Planning for Cooperating UAVs*, pp. 23–41. Springer, 2003.

[70] G. Elghoumari, K. Hila, E. Delechelle, and E. Petit, "Analyse qualitative des images par propagation de front d'ondes," 2001. GRETSI, Groupe d'Etudes du Traitement du Signal et des Images.

[71] J. Sethian, "A fast marching level set method for monotonically advancing fronts," *Proceedings of the National Academy of Sciences of the United States of America, vol 33*, pp. 1591–1595, 1996.

[72] J. Sethian and A. Vladimirsky, "Ordered upwind methods for static hamilton-jacobi equations," *Proceedings of the National Academy of Sciences of the United States of America, vol 98*, pp. 11069–11074, 2001.

[73] S. Vilardaga and X. Prats, "Conflict free trajectory optimisation for complex departure procedures," in *6th International Conference on Research in Air Transportation (ICRAT2014)*, 2014.

[74] S. Vilardaga and X. Prats, "Mass estimation for an adaptive trajectory predictor using optimal control," in *Proceedings of the 5th International Conference on Application and Theory of Automation in Command and Control Systems (ATACCS15)*, pp. 75–84, 2015.

[75] X. Prats, V. Puig, J. Quevedo, and F. Nejjari, "Multi-objective optimisation for aircraft departure trajectories minimising noise annoyance," *Transportation Research Part C: Emerging Technologies*, vol. 18, no. 6, pp. 975 – 989, 2010. Special issue on Transportation Simulation Advances in Air Transportation Research.

[76] X. Prats, V. Puig, and J. Quevedo, "Equitable aircraft noise-abatement departure procedures," *Journal of Guidance, Control, and Dynamics*, vol. 34, pp. 192–203, 2011.

[77] X. Prats, V. Puig, and J. Quevedo, "A multi-objective optimization strategy for designing aircraft noise abatement procedures. case study at girona airport," *Transportation Research Part D: Transport and Environment*, vol. 16, pp. 31–41, 2011.

[78] M. Soler, A. Olivares, and E. Staffetti, "Hybrid optimal control approach to commercial aircraft trajectory planning," *Journal of Guidance, Control, and Dynamics*, vol. 33, no. 3, pp. 985–990, 2010.

[79] D. Toratani and S. Ueno, "A study on trajectory optimization for the terminal area," in *6th International Conference on Research in Air Transportation (ICRAT2014)*, 2014.

[80] D. Toratani, S. Ueno, and T. Higuchi, "Simultaneous optimization method for trajectory and sequence for receding horizon guidance in

terminal area," *SICE Journal of Control, Measurement, and System Integration*, vol. 8, no. 2, pp. 144–153, 2015.

[81] D. Toratani, D. Delahaye, S. Ueno, and T. Higuchi, "Merging optimization method with multiple entry points for extended terminal maneuvering area," in *4th ENRI International Workshop on ATM/CNS (EIWAC2015)*, 2015.

[82] S. Khardi and L. Abdallah, "Optimization approaches of aircraft flight path reducing noise: Comparison of modeling methods," *Applied Acoustics*, vol. 73, no. 4, pp. 291–301, 2012.

[83] J. Betts, "Survey of numerical methods for trajectory optimization," *Journal of Guidance, Control, and Dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[84] O. Khatib and J.-F. Le Maitre, "Dynamic control of manipulators operating in a complex envrionment," in *3rd Symposium on Theory and Practice of Robots and Manipulators*, pp. 267–282, 1980.

[85] D. Kim, H. Wang, G. Ye, and S. Shin, "Decentralized control of autonomous swarm systems using artificial potential functions: Analytical design guidelines," in *43rd IEEE Conference on Decision and Control (CDC)*, 2004.

[86] L. Guys, *Planification de Trajectoires d'Avions sans Conflit : Fonctions Biharmoniques et Fonction de Navigation Harmonique.* PhD thesis, Universite Toulouse 3 Paul Sabatier, 2014.

[87] F. Rejiba, *Modélisation de la Propagation des Ondes Electromagnétiques en Milieux hétérogenes - Application au Radar Sol.* PhD thesis, Université Pierre et Marie Curie - PARIS VI, 2002.

[88] N. E. Dougui, *Planification de Trajectoires Avion : Approche par Analogie Lumineuse.* PhD thesis, Universite Toulouse 3 Paul Sabatier, 2011.

[89] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *IEEE International Conference on Neural Networks*, 1995.

[90] C. Goh, K. Tan, D. Liu, and S. Chiam, "A competitive and cooperative co-evolutionary approach to multi-objective particle swarm optimization algorithm design," *European Journal of Operational Research 202*, pp. 42–54, 2010.

[91] J. Holland, *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975. second edition, 1992.

[92] D. Gianazza, N. Durand, and N. Archambault, "Allocating 3d-trajectories to air traffic flows using a* and genetic algorithms," in *CIMCA 2004, international conference on Computational Intelligence for Modelling, Control and Automation*, 2004.

[93] D. Gianazza and N. Durand, "Separating air traffic flows by allocating 3d-trajectories," in *23rd Digital Avionics Systems Conference (DASC2004)*, vol. 1, pp. 2.D.4–21–13, 2004.

[94] D. Gianazza and N. Durand, "Assessment of the 3d-separation of air traffic flows," in *6th USA/ Europe Air Traffic Management Research and Development Seminar (ATM2005)*, 2005.

[95] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[96] Q. Zhang and H. Li, "Moea/d: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, 2007.

[97] V. Ho-Huu, S. Hartjes, H. Visser, and R. Curran, "An efficient application of the moea/d algorithm for designing noise abatement departure trajectories," *Aerospace*, vol. 4, p. 54, 2017.

[98] V. Ho-Huu, S. Hartjes, L. Geijselaers, H. Visser, and R. Curran, "Optimization of noise abatement aircraft terminal routes using a multi-objective evolutionary algorithm based on decomposition," *Transportation Research Procedia*, vol. 29, pp. 157 – 168, 2017. Aerospace Europe CEAS 2017 Conference.

[99] V. Ho-Huu, S. Hartjes, H. Visser, and R. Curran, "Integrated design and allocation of optimal aircraft departure routes," *Transportation Research Part D: Transport and Environment*, vol. 63, pp. 689 – 705, 2018.

[100] V. Ho-Huu, S. Hartjes, H. Visser, and R. Curran, "An optimization framework for route design and allocation of aircraft to multiple departure routes," *Transportation Research Part D: Transport and Environment*, vol. 76, pp. 273 – 288, 2019.

[101] J. Zhou, S. Cafieri, D. Delahaye, and M. Sbihi, "Optimization of arrival and departure routes in terminal maneuvering area," *6th International Conference on Research in Air Transportation (ICRAT 2014)*, 2014.

[102] J. Sethian, "Fast marching methods," *SIAM Review*, vol. 41, pp. 199–235, 1998.

[103] Y. Huo, D. Delahaye, J. Ma, and M. Sbihi, "Integrated Traffic Flow Based Optimization of Airport and Terminal Area," in *SID 2019, 9th SESAR Innovation Days*, 2019.

[104] M. Liang, *Aircraft Route Network Optimization in Terminal Maneuvering Area*. PhD thesis, Université Toulouse 3 Paul Sabatier, 2018.

[105] K. Liu and M. Zhang, "Path planning based on simulated annealing ant colony algorithm," *9th International Symposium on Computational Intelligence and Design*, 2016.

[106] M. Kobilarov and G. Sukhatme, "Near time-optimal constrained trajectory planning on outdoor terrain," in *IEEE International Conference on Robotics and Automation*, IEEE.

[107] R. Geraerts and M. Overmars, "A comparative study of probabilistic roadmap planners," *Algorithmic Foundations of Robotics V, Springer Tracts in Advanced Robotics vol 7*, pp. 43–57, 2004.

[108] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," tech. rep., Computer Science Department, Iowa State University, 1998.

[109] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," *Robotics Science and Systems VI*, 2010.

[110] J. Gammell, S. Srinivasa, and T. Barfoot, "Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014)*, pp. 2997–3004, 2014.

[111] J. Gammell, S. Srinivasa, and T. Barfoot, "Bit*: Batch informed trees for optimal sampling-based planning via dynamic programming on implicit random geometric graphs." 2014.

[112] J. Gammell, S. Srinivasa, and T. Barfoot, "Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," *IEEE International Conference on Robotics and Automation (ICRA 2015)*, pp. 3067–3074, 2015.

[113] J. Gammell, S. Srinivasa, T. Barfoot, S. Choudhury, and S. Scherer, "Regionally accelerated batch informed trees (rabit*): A framework to integrate local information into optimal path planning," *2016 IEEE International Conference on Robotics and Automation (ICRA 2016)*, pp. 4207–4214, 2016.

[114] T. A. Granberg, T. Polishchuk, V. Polishchuk, and C. Schmidt, "Automatic design of aircraft arrival routes with limited turning angle," *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS16).*, 2016.

[115] V. Polishchuk, "Generating arrival routes with radius-to-fix functionalities," *7th International Conference on Research in Air Transportation (ICRAT 2016)*, 2016.

[116] Eurocontrol, *Guidance Material for the Design of Terminal Procedures for DME/DME and GNSS Area Navigation*, 1999.

[117] Eurocontrol, *Guidance Material for the Design of Terminal Procedures for Area Navigation (DME/DME, B-GNSS, Baro-VNAV and RNP-RNAV)*, 2003.

[118] ICAO, *Required Navigation Performance Authorization Required (RNP AR) Procedure Design Manual*, 2009.

[119] B. Schäffer, C. Zellmann, S. Pluess, K. Eggenschwiler, R. Bütikofer, and J. Wunderli, "Sound source data for aircraft noise calculations - state of the art and future challenges," in *EURONOISE 2012*, 2012.

[120] "Aircraft noise levels." https://www.faa.gov/about/office_org/headquarters_offices/apl/noise_emissions/aircraft_noise_levels/.

[121] O. Laval, S. Toulouse, and A. Nagih, "Rapport de recherche sur le problème du plus court chemin contraint," tech. rep., 2006.

[122] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.

[123] D. Delahaye, S. Chaimatanan, and M. Mongeau, *Simulated Annealing : From Basics to Applications*. Springer, 2018.