UNIVERSITY OF CALIFORNIA
SANTA CRUZ

A GEOMETRIC COMBINATORIAL SATISFIABILITY APPROACH TO AUTOMATING FREE FLIGHT IN THE AIRSPACE

A dissertation submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER ENGINEERING

by

James W. Herriot

December 2023

The Dissertation of James W. Herriot
is approved:

Professor Gabriel Elkaim, Chair

Professor Adam M. Smith

Professor Bruce Sawhill

Peter Biehl
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

A Geometric Combinatorial Satisfiability Approach to Automating Free Flight in the Airspace

by

James W. Herriot

This dissertation details a research project whose aim is to contribute to research on automating and optimizing the aggregate trajectories of arriving aircraft flights in the terminal airspace inspired by the principles of free flight. A software simulation was designed and developed to model dynamic generation of airspace flight paths from top-of-descent to landing. Although this research is conceived to apply generally to automatically generating safely separated flight paths in the wider airspace, this work focuses on airspace local to a particular airport with one or more runways available for landing arriving aircraft. This approach is robust within a dynamic airspace which is frequently changing due to weather, runway reversals, miss approaches, and aircraft traffic.

The methodology described here employs a trio of geometry, combinatorics, and satisfiability. The unique uniform triangular/hexagonal geometry enables the generation of a large number of similar flight path segments (analogous to pixels), which when joined together form the vast combinatorics of all possible 4DT flight paths for aircraft headed to these runways. In this combinatoric form, these numerous flight path candidates are then handed off to a satisfiability solver, searching for a suite of overall optimal flight paths. This approach generalizes solving for an ensemble of efficient flight paths within a dynamic changing airspace, making way for efficient optimization and automation of aircraft routes.

To my parents Drs. Jack and Sally Herriot, my children Liv and Neil Herriot, my partner Dr. Carolyn Arnold, and my esteemed mentors, colleagues and co-adventurers in the worlds of mathematics, philosophy, computer science, engineering, combinatorics, aviation, in chronological order: Prof. Donald Knuth, Dr. Jonathan Ketchum, Prof. Peter Putnam, Dr. Stuart Kauffman, Prof. Bruce Sawhill, Mr. Michael Brown, Mr. Edward Iacobucci, Dr. Marty Klein, Mr. William Bolden, Prof. Gabriel Elkaim, Prof. Renwick Curry, Prof. Adam M. Smith.

Acknowledgments

Modern aviation presents a surprising number of difficult multi-faceted unsolved challenges. I have been fortunate to have had many teachers, friends, family, as well as pilots and strangers on airlines, peering out the same window together, ruminating on my endless questions, and help guide by thinking.

I am especially fortunate to have had the opportunity to co-found an airline, DayJet, as well as serving as a primary investigator on NASA research projects. Both of these efforts involved collaboration and inspiration from my longtime colleague and friend, Dr. Bruce Sawhill. There, I have been able to experience firsthand many of the intricacies, ideas, pleasures, and challenges of modern aviation.

Beyond aviation, I believe many of the ideas and the approach used here apply well to other seemingly disparate fields. I have a long standing personal academic interest in computational modeling of cognitive processes, which I plan to continue to pursue after this Ph.D. project. My research here has deepened by understanding in a broad range of fields, including the specific issues in aviation discussed in these pages.

I thank my committee, Professors Gabriel Elkaim, Bruce Sawhill, Adam M. Smith, Renwick Curry, without whose efforts, generous time and thoughtful attention, this project would not have been possible.

I am grateful to the University of California at Santa Cruz for its continued leadership as a great research and teaching institution, and in particular The Autonomous Systems Lab, founded and led by Prof. Gabriel Elkaim.

Chapter 1

Introduction

This thesis details the key ideas, algorithms, computational simulator inner workings, and results of this project aimed towards automating the airspace using a discrete geometric combinatorial approach. This is a "systems" thesis, centered around the construction and operation of a computational simulator platform to explore the goal of automating the airspace under dynamically changing flight conditions.

This project grew out of my co-founding DayJet airline in Florida in 2002, the world's first per-seat on-demand airline. Becoming immersed in the many facets of aviation through this practical experience led me to a firsthand view of many unsolved problems in aviation. One of those problems is how to improve flight operations in the most congested, least efficient, terminal portion of aircraft flights. This research builds upon the many prior efforts to reduce flight costs through better aggregate trajectory optimization, including my own [86] [87] [43] [44] [53].

In order to concentrate on the most important segment of typical aircraft flight (as measured in operational inefficiency) as well as to reduce the problem scope here to a specific flight phase, this work is limited to arrival traffic, specifically, an adjustable approximately 200 nautical mile radius circle is scribed around a destination airport. This abstract circular area defines the “workspace” or local airspace used in this simulation project. Aircraft may enter this local circular airspace through its perimeter from any compass direction at any time. Once entered, the goal is to ”fly” the aircraft to an airport runway threshold along an optimal 4DT flight path (in the context of other flight paths), while always maintaining safe flight-path separation of all aircraft currently flying in the workspace, respecting dynamic constraints such as airspace closures, convective weather, missed approaches, and of course other traffic.

1.1 Motivations

My professional experience in aviation has included co-founding a small airline and working as a co-principle investigator on various research projects for NASA (the first “A” in NASA stands for “Aeronautics”). At NASA, I observed an intriguing tension between the desire to automate the airspace, and the institutional headwinds valuing fastidious human control to ensure safety. My take-away is that excellent research and careful simulations are needed for each step in automating the airspace of the future. The overall contribution of this thesis research is to further automated airspace research though novel ideas coupled with simulation and computational modeling of safe new ways to automate the airspace. A set of more detailed list of the seven thesis contributions of this research is enumerated in Section 1.6 below.

The salient current problem is that domestic FAA (Federal Aviation Administration) ATC (Air Traffic Control) system or NAS (National Airspace System), although remarkably safe, does not scale well. There are about 14,000 FAA air traffic control specialists who handle about 45,000 flights per day in the US domestic airspace [40]. Automation of the airspace promises improved scaling of the air traffic control system. This question is: how to automate the airspace, while retaining safety, in particular the safe separation of aircraft aloft and upon landing. Airspace automation also promises increased efficiency. Current arrivals flight paths are rigidly defined (see STAR procedures discussed in Section 1.5.4), making them often longer and more circuitous as compared to hypothetical more direct flight paths, or *free flight*, where free flight is flying as if one has the entire airspace to oneself.

The STAR (Standard Terminal Arrival procedures) paths are typically longer than free flight would be. On top of that, the current actually flown flight paths are often even longer than the STARs aviation charts would indicate. This is due to deviations from canonical flight paths used to slow down traffic to defend against congestion which is exacerbated by the limited number of landings (operations or "ops" per unit time) allowed at the runway thresholds. See illustration of an artist's conception of necessary detouring (or vectoring) off the canonical STARs pre-designed flight path, depicted in Figure 1.1, where the blue line is an example of a detour.

The aggregate behavior of airspace traffic congestion together with ATC adhoc flight path change interventions creates a kind of "scribbling" on flight tracks radar records of actual flight-path signatures. See illustration in Figure 1.2.

The motivation for the research described in this thesis was to explore a possible design of an automated airspace which delivers improved operational precision and

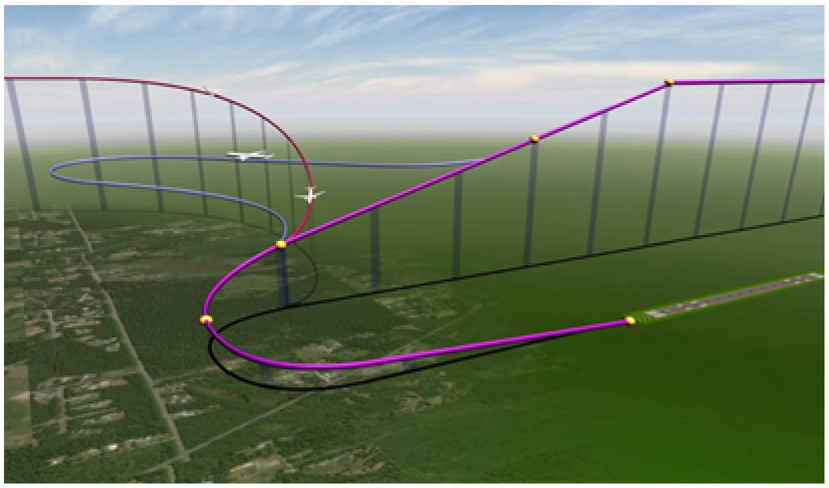


Figure 1.1: Artist's conception for visualization of 2 pre-defined fixed arrival flight paths, blue detour illustrates "vectoring" of an aircraft. [71]

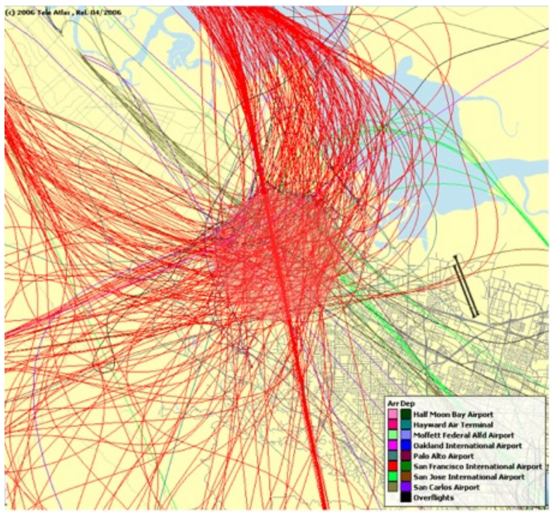

(c) 2006 Tele Atlas , Rel. 04/2006
Arr Dep
Half Moon Bay Airport
Hayward Air Terminal
Moffett Federal Afld Airport
Oakland International Airport
Palo Alto Airport
San Francisco International Airport
San Jose International Airport
San Carlos Airport
Overflights

Figure 1.2: Radar flight tracks from June 2, 2014 superimposed showing evidence of extensive vectoring or non-direct paths into SFO over the Palo Alto area. [92] This graphic was drawn from FAA radar data, obtained through a FOIA. [17]

efficiency, while retaining safety levels of the current manual ATC system.

What would such an automated free-flight-like airspace look like? Would aircraft continue to take the routes they already follow today? Or would aircraft take routes analogous to water molecules spiraling down a bathtub drain? Or would the flight paths be unexpectedly different from either of these possibilities?

How should such an airspace (simulation) be constructed? Can methods drawn from fields outside of aeronautics fields contribute to helping solve these problems? Can a novel airspace geometry be combined with recent developments in combinatorics and satisfiability solvers help make substantive progress in this aeronautical domain?

I was motivated to follow these questions toward a demonstrable, suggestive solution. This thesis reports on this project.

1.2 Free flight and optionality

The above Section 1.1 on Motivations alludes to "free flight". This is an important concept here, so some additional clarification is indicated.

The notional opposite of rigidly prescribed navigational flight route procedures (see Subsection 1.5.4) is *free flight.* This is a relaxation of navigational protocols. Hence, pilots are free to take any safe route they choose, at least in principle. A typical definition of free flight is as follows:

> Free flight is defined as safe and efficient flight operations under instrument flight rules in which the operators have the freedom to select their path and speed in real time. Thus, individual aircraft will be able to plan and execute their trajectories without direction from any external agents during most of their flight duration. [72]

This research project was predicated on the possibility that the principles of free flight, as incorporated into automated flight control, might improve efficiency of aircraft descending in terminal airspaces, as well as relieving some congestion, while continuing to maintain safety in the airspace.

The intuition here is that by relaxing the rigid navigational rules of STARs dictated flight paths, the aircraft would have increased routing optionality. This suggests that many shorter, more optimal routes, restricted by STARs would become available to fly. Hence, as more shorter paths are added to the pool of possible flight paths, the length of the average flight path would become shorter, and hence closer to optimal.

Figure 1.3 shows a notional visualization of a comparison between a STARs style restricted path, and a more direct free flight style path. A glance at this schematic uncalibrated example suggests that a path length savings of 20% might be plausible. Note that it is beyond the scope of this thesis to determine the average improvement in path length through the use of automated free flight.

In the current system, free flight would be seen as too risky within the "super dense" airspace of arriving flights at busy airports. However, this thesis takes the position that automation can indeed defend against these risks. Even "super dense" airspace can be safely automated using some of the principles of free flight.

This is not to say that the pilots themselves would be free to choose their own routes. Rather, the pilots would still be required to fly the routes as instructed by the automated flight control system, but these routes would be generated dynamically much as if the pilots chose the most efficient paths to a runway, but with full knowledge and cooperation of the other pilots' plans.

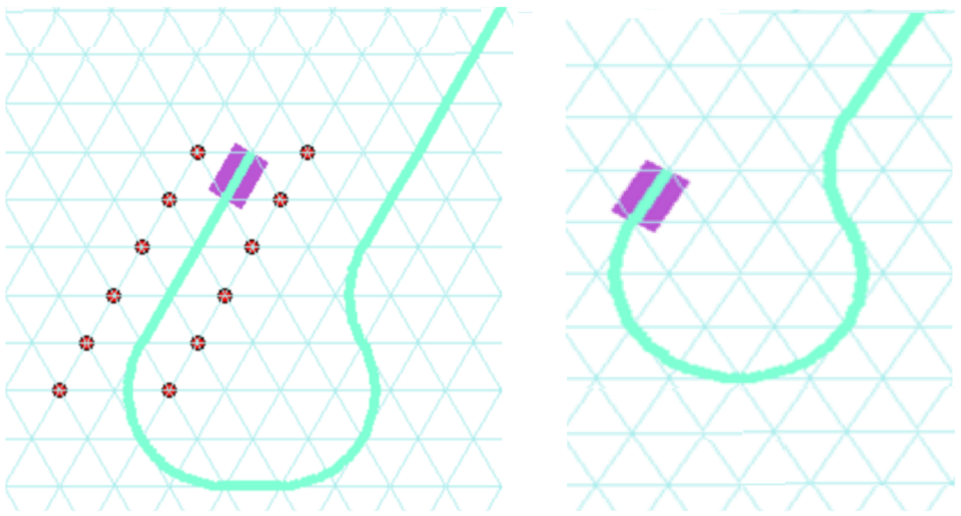


Figure 1.3: Notional comparison of a restricted STARs-style flight path (left graphic) versus a shorter free flight style path which takes the shortest route (right graphic) to a runway (purple).

1.3 Combinatorial approach

Chapter 2 provides a system overview description of the proposed combinatorial approach towards an automated system for optimally landing an ensemble of airliners, guiding them from cruise altitude to landing at a runway at a designated destination airport.

This means that large numbers of candidate flight paths for the full ensemble of aircraft are evaluated, and the overall most optimal set of flights are chosen to be flown for next period of time. Later Chapters 5 and 6 discuss the details of how that optimal set of flights is calculated, based on the geometry detailed in Chapters 3 and 4.

Only the arrivals terminal portion of flights is addressed here. This system is based on free flight (discussed in Section 1.2) above and employs a scalable, efficient 'holistic' way (aircraft treated as a single group) to manage the local terminal portion of the airspace. This approach may generalize to the wider airspace including the NAS

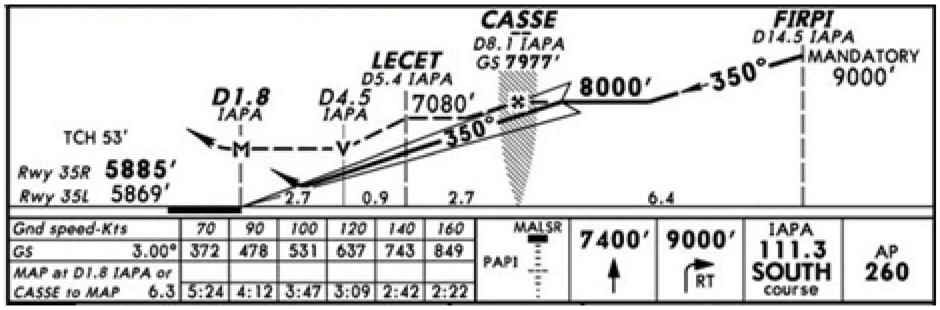
(National Airspace System) and beyond, as well as specialized custom-use airspaces, but that is beyond the scope of this research.

1.4 History

In 1921, Croydon Airport, London's main airport at the time, was the first airport in the world to introduce air traffic control [29]. In the USA, 5 years later, President Coolidge signed the Air Commerce Act of 1926 into law [36]. In 1929, Air Traffic Control (ATC) began with Archie W. League at the St. Louis airport. His communication tools and protocol were simple: a red flag for "hold" and a checkered flag for "go."

The first aeronautical charts were produced in 1941, supporting first instrument approaches. Aviation advanced quickly, especially so during World War II, with the advent of radar (an acronym for radio detection and ranging). By 1952, radar was routinely used for approach and departure control [76]. See example of a sample of a modern aeronautical chart depicted in Figure 1.4.

The Federal Aviation Administration (FAA) was founded as part of the Federal Aviation Act of 1958, in the aftermath of a tragic mid-air airliner collision over the Grand Canyon in 1956, depicted in Figure 1.5.. Since that time, specific routes – especially terminal arrival routes – have been engineered. The standard set of such routes, called standard terminal arrival routes ("STARs"), are used by the FAA ATC for air controllers to guide and supervise aircraft approaches to airports – for both commercial and general aviation uses.



Gnd speed-Kts	70	90	100	120	140	160
GS 3.00°	372	478	531	637	743	849
MAP at D1.8 IAPA or CASSE to MAP 6.3	5:24	4:12	3:47	3:09	2:42	2:22

Figure 1.4: Aeronautical Chart used by pilots to navigate according to a predefined Standard Terminal Arrival Route (STAR). This shows the vertical profile of a STAR descent into SFO. When combined with top-view geographical maps, and radio contact with ATC, pilots can successfully and properly fly a designated STAR. [39]

Figure 1.5: Artist conception of the tragic mid-air airliner collision over the Grand Canyon in 1956 that prompted the founding of the Federal Aviation Administration (FAA). [47]

Safety has been the primary concern of the FAA since its founding to the present day. The FAA focuses on safety, while paying less attention airline costs, which are primarily time and fuel, although this may be slowing changing. [38]

With safety taking precedence over airline costs and environmental factors, the current Air Traffic Control (ATC) system has evolved to be impressively safe though inefficient and with a secondary focus on the environment. To insure this safety, the ATC is structured as a command and control system. To this end, the system employs a small set of pre-defined fixed flight paths, monitored manually, for the terminal or final descent phase of arriving flights. For large airports, there are typically 4 main fixed flight paths per airport called Standard Terminal Arrival Routes (STARs, see more in section 1.5.4) for any particular configuration (tailored to wind direction) nominally serving the 4 arrival compass directions. Aeronautical charts defining STARs are available from the FAA, such as the one in Figure 1.4 [39].

These STAR fixed flight paths can be visualized as "tubes" in the sky. Ideally, pilots program their Flight Management Systems (FMS) to follow these routes as specified by the FAA. However, congested air traffic often forces the ATC to "vector" traffic temporarily off the specific routes to add time to the descent to enforce aircraft separation and spacing. Figure 1.1 is an artist's conception of these tube-like routes, as well as a "vectored" detour shown in blue.

The neat pipe-like routes depicted in Figure 1.1 with an occasional "vector" detour, together with the schematic navigational charts (e.g. Figure 1.4) convey an orderliness to the arrival descent process. In reality, the actual flight tracts are far from this orderly ideal. As shown in Figure 1.2, the flight tracks display a large variation from the ideal, due to extensive vectoring.

1.5 Prior Art

1.5.1 General Problem

There is an extensive research literature in motion planning, as well as its subdiscipline, vehicle motion planning for ground-based robots and UAVs. These problems are known to be “difficult to solve”.

> There does not exist an algorithm that provides an exact analytic solution to such a problem... Indeed, even state-of-the-art approximation algorithms operating on a three-dimensional subspace of this problem space are difficult to compute in real time. [45]

Nevertheless, much progress has been made in this class of problems. To do so, the general problem area has been divided to a number of more tractable sub-areas. Problems are typically divided into static and dynamic, where dynamic includes both imperfect knowledge of the environment, as well as a changing environment [45]. These problems are divided into flat (2D), 3D, or full 4DT solution spaces.

1.5.2 Aviation

The focus of this work is on aviation here; not ground based robots. Within aviation, much of the current research has been for UAVs, as opposed to passenger aircraft. As such, the extent of geographical space is typically on the order of 100s of meters, as opposed 1000s of kilometers typical for commercial aviation. Distance is typically the primary metric for UAVs due to their limited endurance. In commercial aircraft, the primary metrics are time and fuel.

For UAVs, changing environments can include avoiding obstacles that are themselves moving over time. This can be formalized as a continuous mathematics problem. [56] However, adequate compute performance can be an issue here.

Alternatively, the dynamic problem with a changing environment can be conceived and solved as a discrete problem, represented as a graph, where the nodes or vertices have geographical semantics. This has been done in the UAV problem space [18], but relatively short distances and times are typically optimized here, not to mention ignoring fuel consumption per se.

1.5.3 Air Traffic Control

As section 1.4 on history discusses, aviation progressed quickly from the 1920s till the founding of the FAA in 1958. Although a previous Federal government agency, the Civil Aeronautics Authority (later renamed the Civil Aeronautics Board), had been operating since 1938 [76], the advent of the FAA hastened changes in Air Traffic Control.

More changes came quickly. The "Jet Age" for commercial aviation also began with the first Douglas DC-8 flight on May 30, 1958 [36]. New procedures were needed to manage the increasing amount of air traffic. For example, Standard Instrument Departure (SID) procedures first went into effect on August 13, 1961 [36]. Standard Terminal Arrival Route (STAR) procedures went into effect at a similar time, although the exact date is illusive.

The literature in aviation suggests that prior to the 1960s, most innovations in Air Traffic Control came from federal agencies. Academic research appears to follow to measure, analyze, and improve the basic design of the airspace put in place by government authorities. There is an early reference to the Journal of Air Traffic Control in 1964 [16], an indication that Air Traffic Control became an area of academic investigation in the era after the advent of SID and STAR procedures.

1.5.4 STAR procedures

Currently, normal aircraft descents in the terminal airspace are required to follow a Standard Terminal Arrival Route (STAR). These are fixed arrival routes (3D paths) used by the FAA and ATC. They are of interest in this thesis for notionally comparing fixed STAR-like arrival routes with the simulation of guided free flight in this project.

STAR procedures came into being, along with Standard Instrument Departure (SID) procedures in the 1960s [76]. They are a result of new technical capabilities and perhaps also hastened by the collision of a United DC-8 and a TWA Super Constellation in terminal airspace over Brooklyn on December 16, 1960 [36].

As important as STARs are for this research, in the history of aviation, they are just one of many *procedures* required by the FAA. These various procedures are typically artifacts of ever new technical capabilities (e.g. radar monitoring and radio communication, ADS-B i.e Automatic Dependent Surveillance–Broadcast [41]) and necessity, rather than academic research. Once, these procedures were in place, academic research tended to follow to investigate particular phenomena and to improve safety performance.

Within the field of aviation research, the literature on Air Traffic Control has grown from the early days of the new procedures [75]. One area of focus is on improving the current system of STARs managed by ATC [43] [44] [86] [87]. There is much to be improved there, and large monetary savings to be harvested from this type of research. In contrast, although research into free flight goes back to the 1990s and before [72], the current systems are expected to last for years to come. The literature tends to be abundant addressing incremental improvement in the current system, while the literature addressing free flight is more sparse.

1.5.5 Path Stretching

A discussed in Section 1.1 above, the current system with fixed path STAR procedures channels a sequence of aircraft descending single file (like a “Conga line”) down narrow flight corridors. One of the consequences of this system is that aircraft following each other can become closely packed. So, in order to avoid separation conflicts, ATC frequently instructs aircraft to detour off the STAR briefly, which may lengthen its path to the runway, depicted in Figure 1.1. Hence, that aircraft’s path is “stretched”. This path stretching is also called “vectoring”. Estimates are that this can result in approximately 7% longer flight paths [88].

One of the consequences of the current system with fixed path STAR procedures is frequent ad hoc path stretching by ATC, with its corresponding extra costs. This problem has been studied extensively [89]. It is also projected to become a more pressing problem as air traffic increases over the next 20 years [74]. Various attempts have been made to improve the situation [105].

1.5.6 Discrete Solutions

An alternative to continuous solutions are approximations in the form of a discrete formulation of the problem. The merit of this approach is typically better computational performance. Naturally, as an approximation, the solution will not be optimal. But the departure from optimal can be made arbitrarily small at the expense of increased computation.

The discrete problem can be mapped to graph theoretic representation [102], where the objective becomes to find the shortest path from origin to destination airport via the edges of a graph. A typical way to form a discrete space from a continuous space is a rectangular grid, or more complex geometries [60].

The most straightforward method to determine the shortest path between 2 vertices is the classic "Dijkstra" algorithm conceived by Edsger W. Dijkstra in 1956 [32]. Scaling is O(V^2). Since that time, improvements in that algorithm have been made by adding heuristics. The best known of these is the A* or A-star algorithm [49], with later improvements as well [83]. Similar algorithms are still in use today, including ones used in this project.

1.5.7 Scaling

Although discrete solutions are widely in use today for transcontinental commercial air routing, these graph-based paths, typically solved by A* algorithms, are not fine grained enough to resolve obstacles as small as a nearby aircraft. The main features of the airspace addressed in those planning operations are wind and weather. Scaling from the 1 nautical mile (nm) resolution all the way to global distances is beyond the reach of current A* planning algorithms in use today. Indeed, comprehensive aircraft routing ultimately requires optimal trans-oceanic routes that avoid large weather systems as well as other relatively small aircraft. One of the future goals of this research is to build algorithms with a breadth of scale from 1 to 1000s of nautical miles.

1.5.8 Satisfiability

Given that flight planning in a discrete space is ultimately a constraint problem, it would seem to be a natural fit to formalize the problem into the terms of Satisfiability or 'SAT" [61] [12] [63] [20]. However, the lack of articles on a SAT approach to flight planning in the academic literature suggests that this may be a methodology too foreign to aviation researchers, or too computationally intensive to be practical, at least at the present time (even though this may no longer be the case with modern satisfiability solvers, and lower computation costs).

Indeed in his Fascicle 6 on Satisfiability, Don Knuth notes that SAT can even be used to factor numbers without any appeal to number theory. However, as he says, "we can't expect this method to compete with sophisticated factorization algorithms" [63]. Likewise, a general SAT approach may, in principle, solve arbitrary flight planning problems. However, they may be too slow to compete against specialized approaches.

This thesis takes the position that Satisfiability has great merit as part of a novel methodology for flight path planning in real time.

1.5.9 Closely Related Prior Art

Single-Aircraft Flight Planning

There is a vast literature on single-aircraft flight planning. There also many mature products used by General Aviation (GA) pilots, and others. The research as a whole here is well developed too. Since this field does not address the multi-aircraft airspace treated in this thesis, single-aircraft flight planning prior art is not discussed here.

Multiple-Aircraft Flight Planning

Multi-aircraft (not single-aircraft) planning is addressed in this thesis. This is also a more difficult problem than single-aircraft planning. This difficultly is due to the requirement to resolve path conflicts and cost trade-offs among aircraft sharing the same airspace simultaneously. The resolution of conflicts ultimately determines the actual flown paths as well as overall costs. Indeed, this thesis addresses efficient flight planning for multiple aircraft competing for separated space aloft, with limited time-slots available at runway thresholds for landing.

Multi-aircraft planning can be translated and generalized into multi-agent path planning, where individual aircraft can be mapped to individual agents.

Multiple-Aircraft Converging At Waypoints

By focusing on the narrow problem of focusing on multiple aircraft passage through waypoints, continuous algebraic solutions have been proposed [90]. While this research has value at individual waypoints, it does not address the larger problem of

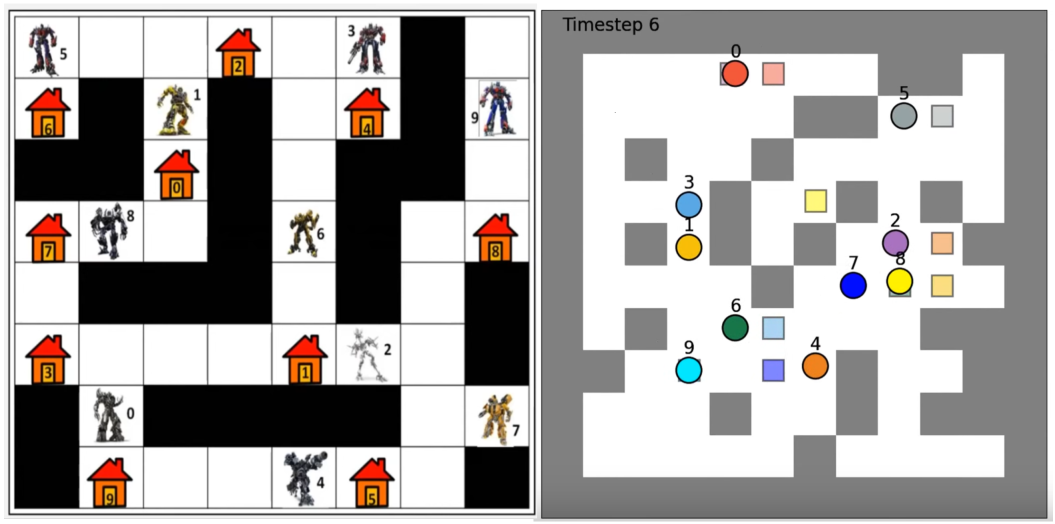


Figure 1.6: Multi-agent path finding (MAPF). A graph with N states, K agents. Constraints: Paths shouldn't conflict. Actions: An agent can move or wait. Target: Minimize the cost of the solution.

aircraft planning from arbitrary positions aloft, descending all the way to a runway. This larger issue is addressed below with various approaches, but with their own limitations.

Multi-Agent PathFinding

Fortunately, there is already a methodology and strong body of literature discussing this generalization of multi-aircraft planning formalized in the literature as Multi-Agent PathFinding or "MAPF". The approach is abstracted as multiple agents moving simultaneously on a discreet rectilinear grid as shown in figure 1.6. [1]

Each agent in each time cycle can move one of 4 compass directions (N,S,E,W), or wait (remain where it is). The two exclusion rules are that no 2 agents may simultaneously occupy the same square or move through the same edge in the same cycle. Figure 1.7 illustrates these rules.

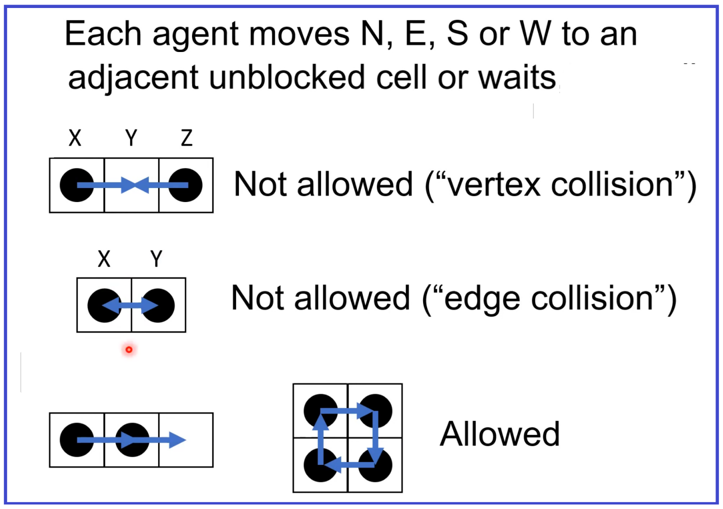

Each agent moves N, E, S or W to an adjacent unblocked cell or waits
X Y Z
Not allowed ("vertex collision")
X Y
Not allowed ("edge collision")
Allowed

Figure 1.7: Multi-agent path finding (MAPF). Each cycle, each agent is allowed to move rectilinearly one step to an adjacent unblocked cell, or wait).

Airport Ground Operations

This MAPF approach has many applications [7]. For example, airport ground operations can be modeled and planned using MAPF as illustrated in Figure 1.8 [2]. The airport taxiways are divided up into squares where aircraft can move or wait as they travel from runways to airport gates, and vice versa.

Pipe Routing

Another application of MAPF is pipe routing [8]. As Figure 1.9 illustrates, aircraft routes can be thought of as *pipes* in the sky, similar to routing industrial plumbing pipes in a factory. The primary difference between aircraft and plumbing pipe routing is the number of dimensions. Whereas plumbing pipes can be expressed in 3D xyz dimensions, the airspace 'pipe' routing requires xyzt 4DT pipes. Hence, airspace routing is a generalization of plumbing pipe routing with one additional dimension.

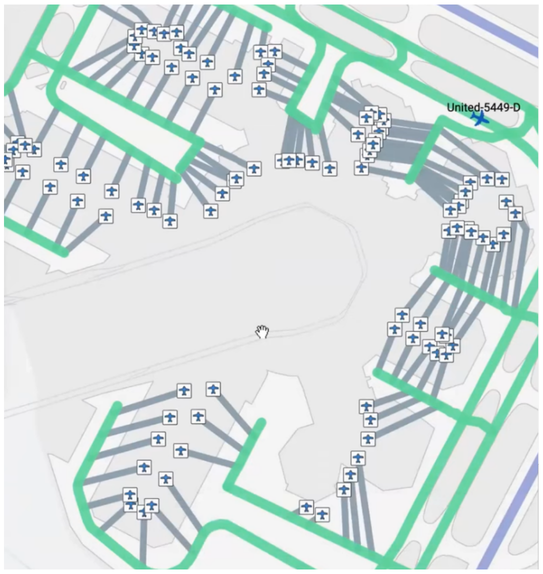


Figure 1.8: Aircraft surface operations can utilize multi-agent path finding (MAPF) to find a non-conflicting way to manage airport ground operations.

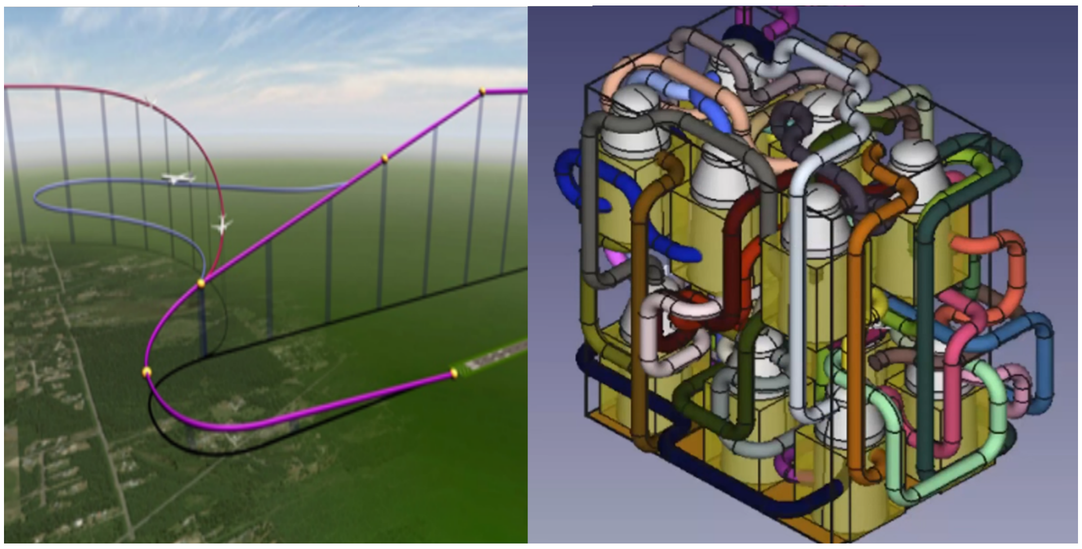


Figure 1.9: Pipe routing examples: left panel shows aircraft flying through 4DT 'pipes' (cf. figure Figure 1.1), right panel shows industrial plumbing pipes in 3D xyz dimensions. For aviation, the pipes concept requires generalizing 3D xyz pipes to xyzt 4DT pipes.

Aircraft Aloft

Applying MAPF to planning and managing multiple aircraft aloft is attractive. However, standard MAPF allows aircraft to move or *wait* each cycle. Whereas aircraft can stop and wait during ground operations, aircraft must maintain some minimum speed while aloft. Hence, MAPF is less attractive for aviation aloft. The equivalent of waiting aloft is for Air Traffic Control (ATC) to *vector* or detour aircraft off their planned route and back on it to stall for time, as illustrated in the left panel of Figure 1.9. This additional complexity, or similar, make standard MAPF less attractive for use in the multiple aircraft planning addressed in this thesis.

NP Hard Problems

The MAPF approach often generates NP hard search problems. Indeed, MAPF problems can be reduced to Satisfiability (SAT) problems which are well known NP hard problems. [9] As with many NP hard problems, there are various approaches to reducing the combinatorics by harvesting structure in the problem, or by settling for non-optimal solutions.

Prioritized Planning

One common way to defend against overwhelming combinatorics in MAPF problems is to use a technique called "prioritized planning", where the agents are solved one agent at a time. Depending on the problem, this technique can give excellent results, but is not guaranteed to do so. [68] [5] [11]

Indeed, the alternate method to pure satisfiability for solving the airspace rapidly is described in this thesis in Section 6.7 on the Custom Solver used here employing "prioritized planning". Excellent computational performance is traded off against best solutions, yielding real-time performance.

UNSAT

MAPF is a particular approach to a wide variety of constraint problems. As with this class of problems, there is frequently the possibility of no solution, or unsatisfiability (UNSAT). This issue is also treated in the MAPF literature. [5], and in Section 7.2.4 on Airspace Capacity, Phase Transitions, Optionality, and UNSAT.

Conflict Avoidance

Matthew Jardin in his Ph.D. dissertation at Stanford [57], did some pioneering research in aircraft conflict avoidance through dynamic path re-planning on a small scale. The work presented in his thesis on generating dynamic flight paths addresses separation conflicts in the airspace at a small enough scale where individual aircraft avoidance is an important issue. In this sense, the work reported in this thesis has some small similarities to part of the work of Matthew Jardin.

Other Techniques

There are also other techniques for improving computational performance in the face of large combinatorics. [1] [67] [66] [7]

1.6 Thesis Contributions

This thesis makes a number of contributions to aviation and air transportation systems. Although it focuses on the arrivals portion of airliner flights, many of the principles, approach, and contributions here apply to entire flight missions, as well as other less conventional airspaces. For example, the dynamical character of the algorithms developed and employed here may apply well to other air transportation systems as disparate as special purpose UAVs, and to airspaces with mobile airports like aircraft carriers or ad hoc airports for emergency humanitarian missions.

The following is a brief list of 7 thesis contributions. Details are discussed in the following chapters below.

- Methodology for operating a fully automated airspace

  The specific approach discussed in this thesis is discreet and possibly non-polynomial hard, as opposed to continuous or MAPF with prioritized planning, etc.

- Uniform geometry for representing flight paths

  The goal was to simplify the software design though a simple geometry that could be replicated throughout the airspace.

- Geometry bonus: flyability, separation, sequencing

  The goal was to leverage the geometry so that as much as possible the aircraft navigation constraints would be accomplished passively through the geometry as opposed to use of active computation to enforce constraints.

- Flight paths constructed Lego-style from path fragment ‘pixels’

A constructionist approach was employed to generate full flight paths by turning path fragments on or off much like pixels form detailed graphics just by turning selected pixels on or off.

- Combinatorics and satisfiability used to find optimal flight paths

  A combinatorial approach begins with the high-level abstraction of enumerating all the possible flight paths in principle, so that the problem is reduced to choosing the best of already known flight paths.

- Frequent dynamic re-planning adapting to the dynamic airspace

  The airspace conditions change over time due to weather, air traffic, availability of runways, etc. Hence, the system needs to replan in the face of these changing airspace conditions.

- Animated graphics simulator for researching this approach

  Visualizations through animated graphics can provide understanding and valuable insight into a system managing a complex airspace with multiple aircraft.

1.7 Thesis Structure

The research described and discussed in this dissertation center around a detailed simulation of the local arrivals airspace for busy airports in the National Airspace System (NAS). This dissertation is structured around explaining the details of a novel approach to guiding aircraft on the this arrivals portion of aircraft flights. In the broadest sense, the two main parts of this dissertation are a detailed description of the system and its algorithms, followed by a discussion of conclusions, and future work.

The structure of this thesis presented here in 7 chapters is as follows:

1. Introduction, motivation, overall problem area and context
2. System Description of the overall research project presented here in this thesis
3. Geometry addressing the two xy dimensions of geography as a triangular grid
4. Geometry in full xyzt 4DT form, representing the structures underlying flight paths
5. Combinatorics methodology for enumerating the vast number of possible flight paths
6. Satisfiability methodology to calculate optimal flight paths for a suite of aircraft
7. Conclusion and future work including the possibilities, limitations of this research

Chapter 2

System Description

2.1 Introduction

The ideas presented in this thesis are aimed at safely automating the airspace. Ultimately these ideas would need to be tested live with real aluminum etc. aircraft in the real airspace. Since that type of testing is neither practical nor safe, a simulation is used here to validate these concepts using computational modeling.

The system described in this chapter is a simulated discrete local arrivals airspace populated with incoming flights (i.e. aircraft) arriving from all over the world, as shown in Figure 2.1. Sample historical FAA data are used generate plausible flights, complete with flight numbers, originating airports, and aircraft type.

The origin airport is only used for its geographical location. The origin airport location is used to determine the approximate compass heading of entry into the local airspace at its perimeter. The direction of entry is defined by the great circle route from the origin airport to the airport destination at the center of this local airspace,

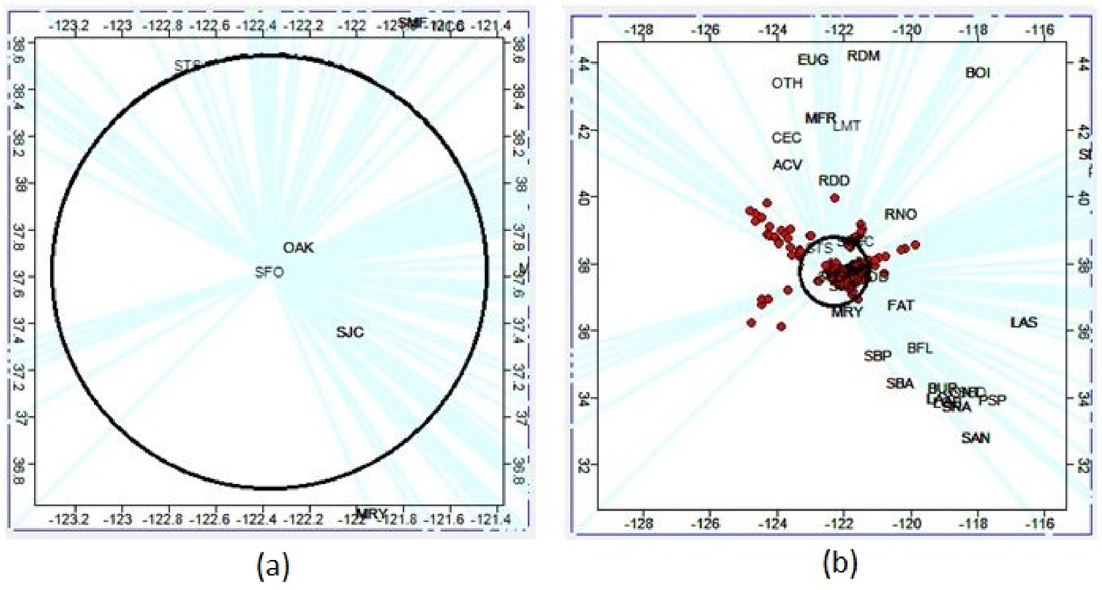


Figure 2.1: Local airspace perimeter drawn with black circle in left graphic with 3 destination airports labeled. The two graphics are at different scales with axes labeled with longitude and latitude. The right graphic shows arriving aircraft labeled with the code of their origin airports. IATA airport codes are used in both graphics. The flight data is drawn from historical FAA data.

independent of weather and winds aloft which would normally affect actual flight routes. Figure 2.2 illustrates 34 perimeter entry triangular edges i.e. line segments, for this particular size of workspace. All entering aircraft into the simulated local airspace enter through one of these line segments on the periphery. The particular angles of the entry line segments reflect the geometry discussed in Chapter 3.

2.2 Simulator

Currently, the proposed system is a computational model only, a software simulation of this free-flight approach to automating the management of arriving aircraft into some number of runways at a single destination airport. It is hoped that some of

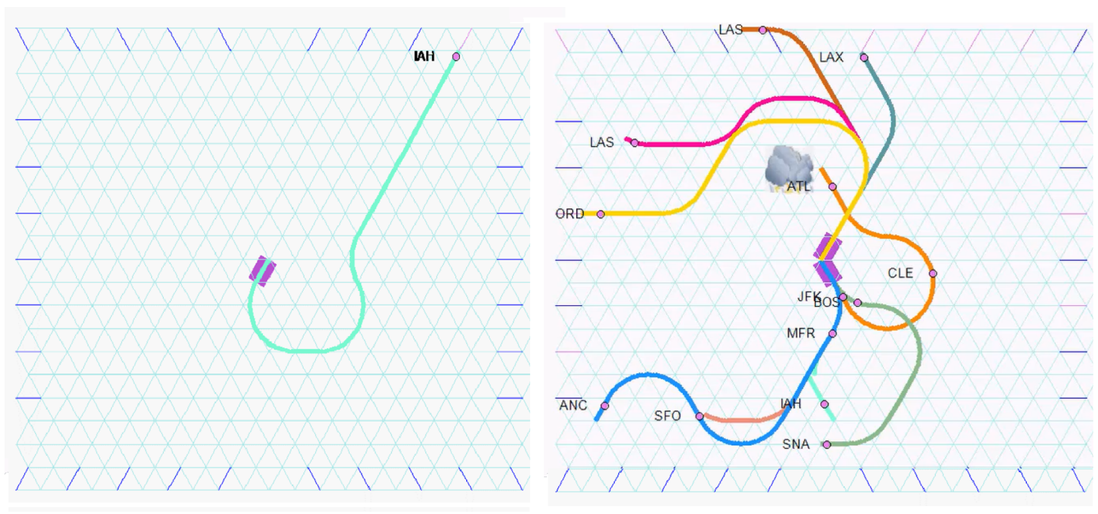


Figure 2.2: Simulator workspace for modeling flights and flight paths. In both graphics the workspace is represented by the same confined triangular grid, with entry points drawn as short blue line segments o the periphery. The left graphics shows a single flight path descending into one runway in the center, while the right side shows multiple flight paths descending into 2 runways of the same airport, as well as a gray convective storm to be avoided as well. Flights and their flight path origins are labeled with the origin airport IATA code. Scale is 143 nautical miles (nm) on each side.

the ideas in this thesis will eventually mature beyond software modeling, and contribute to live deployed air traffic management systems of the future. Clearly, operational real world systems are far more complex than simulations, including the one detailed here. However, the goal here is to demonstrate the key innovative ideas as a first step toward a proof of concept of a live operational system.

The simulator used in this research was conceived, designed, and programmed by the author. The simulator is an interactive agent-based modeling system with an emphasis on adequate physical realism at the scale of flight paths, widely available high-end-PC performance computation, and animated graphics. It can be difficult to imagine the details and mechanics of a suite of aircraft in a dynamically changing 4-dimensional (4DT) airspace. To better understand this intricacies of the algorithms described in these chapters, many screenshot visualizations from the graphics generated by the simulator are displayed in figures here.

2.2.1 Work space

The simulator performs its calculations within a workspace, representing a small adjustable geographical portion of the local arrivals airspace, top view, annotated by altitude and time. Figure 2.2 shows 2 examples of a local airspace workspace, 143 nautical miles (nm) on each side. As discussed in Chapter 3 on 2D Geometry, the airspace and corresponding workspace are structured as a triangular grid. Figure 2.2 is drawn with this triangular structure.

2.2.2 Closed airspace and blocking

By representing the airspace workspace as a geometric triangular grid, flights can be represented via a graph theoretic formalism of vertices and edges. Flight paths are reduced and simplified to sequences of edges and their intermediate vertices.

Since all flight paths (at least nominally) pass through vertices, the concept of "closed airspace" can be implemented by representing closed airspace as forbidden vertices.

An important concept in this systems approach to aviation is *blocking*, drawn from operating systems design. In this discussion, if a vertex is *blocked*, no flight paths will be constructed using this blocked vertex. Hence, closed airspace is implemented by use of set of nearby blocked vertices as shown in Figure 2.3. Indeed, the right-hand panel of Figure 2.2 shows a patch of convective weather with flight paths avoiding those

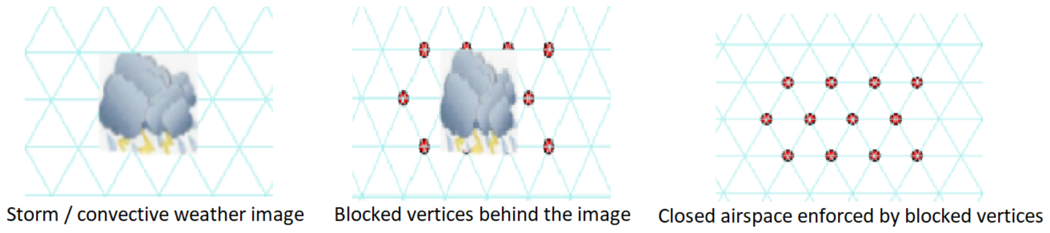

Storm / convective weather image
Blocked vertices behind the image
Closed airspace enforced by blocked vertices

Figure 2.3: Closed airspace, convective weather in this case, implemented by a set of blocked vertices.

closed vertices. Blocking is also used elsewhere in the system as discussed in Chapter 4.

2.3 Dynamic Planning

Flight planning would be easy if it weren‘t for continually changing flight conditions. Convective weather moves unpredictably, runway availability changes, aircraft miss their landings. (See section 4.9 for more on changes in runway directions, etc.) From one minute to the next, the best laid plans may become obsolete. Therefore, timely revised plans are often needed.

2.3.1 Disruptions

More generally, flight planning would be easy if it weren‘t for disruptions – unexpected changes of any sort. One way to operate a system is to arrange for (hope for) a mostly smooth running world, and then address disruptions in an ad-hoc manner as they occur. In this way of thinking, disruptions are a kind of pathology, situations to be avoided, if possible.

However, the proposed system here takes the opposite approach. In this system, disruptions are endemic, and are considered part of a truly robust system. Disruptions include changes in weather, wind affecting runway landing directions, missed approaches, new aircraft entering the local airspace, even aircraft landing and removing themselves from the airspace. In a strong sense, all changes are “disruptions”.

2.3.2 Planning cycles

The proposed system operates as a series of quasi-independent sequential planning cycles, once per minute, in discrete time steps. Each planning cycle manages the local airspace for exactly one minute, planning and flying the aircraft from where it is at the beginning of the minute to where it will be the end of the minute. Each cycle is *stateful* in the sense that the position (x,y,z,t locations), direction and type of the aircraft are retained from cycle to cycle. But each cycle is *stateless* in that all other conditions of the airspace are discarded or “forgotten” between cycles. More precisely, the state of the airspace, apart from the aircraft, is generated stochastically. Hence the proposed system has a relaxed attitude to disruptions precisely because it does not even “know” if there were a “disruption”. It doesn‘t know what attributes of the local airspace remained the same or changed from one cycle to the next. This “amnesiac” or stateless property endows this system with a high degree of generality and robustness.

2.3.3 Replanning

Hence, rather than wait for flight conditions to change, and react in an ad hoc manner, the proposed system replans all the aircraft in the entire local airspace once every minute anyway – regardless of whether flight conditions have changed. Naturally, there is an opportunity for computational optimization when conditions remain the same (discussed more in Chapter 7), but it is more general and simpler to replan every minute regardless of actual need. Also, in the real world, navigation errors are endemic. Such errors are not simulated here, but in the real world, there will no doubt always be at least some value of frequent flight path replanning.

2.3.4 Paradox of long-term planning

The paradox of dynamic flight planning is that the system plans from the current position of the aircraft, all the way to landing at a destination runway, perhaps 20 minutes into the future, but only the first minute is actually used. Issues like lowest cost path, obstacle avoidance, aircraft separation are all calculated for the entire remaining duration of the flight. And yet, one minute later, conditions may have changed, and a whole new path will be re-calculated and replanned. Even though the system diligently plans for the all remaining minutes of the flight, only the first minute of this planning is actually used. The rest is discarded, although it was essential in the planning process used to generate the 1-minute portion of the flight actually used.

The obvious question is why plan 20 minutes ahead, yet only use the first one minute of that plan? The answer is that the best plan for the next minute lies in the context of a longer future. Conversely, if the plan for the next minute had zero ways to fly the rest of the way safely, then it would be a poor choice of a plan. So, the proposed system does extensive planning, yet can react quickly when flight conditions change – as they often do. At the end of each planning cycle, the system “bets” on a single optimal global plan. However, in the succeeding minute, the system might calculate a new different lowest cost global plan, best suited to newly changed conditions.

Perhaps this paradoxical approach to flight path planning is similar to how many people plan their lives: plan what we logistically do next, within the overall context of our long-term life strategy.

2.4 Descriptions of Algorithms

The core of this research are the algorithms conceived, designed, and programmed here, providing the basis for a flexible and powerful simulation a local airspace. As discussed in the Introduction Chapter 1, the facets of the algorithms employed here divide well into three parts: geometry, combinatorics, and satisfiability. This triad of algorithm facets corresponds to the following two Chapters 3 and 4 on Geometry, and the two succeeding Chapters 5 on Combinatorics and 6 on Satisfiability below.

Chapter 3

Geometry - part 1: 2D geography

3.1 Introduction

This thesis details a novel, discrete approach to automating the airspace. To accomplish full automation, at least in principle, a special purpose uniform geometry is proposed here to represent flight paths. This uniform geometry enables a combinatoric (and satisfiability) method for solving a set of optimal flight paths for an ensemble of aircraft descending to an airport.

3.2 Underlying geometry

The geometry proposed here is a uniform triangular grid, illustrated in Figure 3.1, which has a number of useful properties discussed below. This simple pattern of triangles is not itself flyable due the sharp angles at the intersections of edges at the vertices. However, with the addition of some specialized inscribed curves, the geometry becomes flyable, and useful for combinatoric analyses.

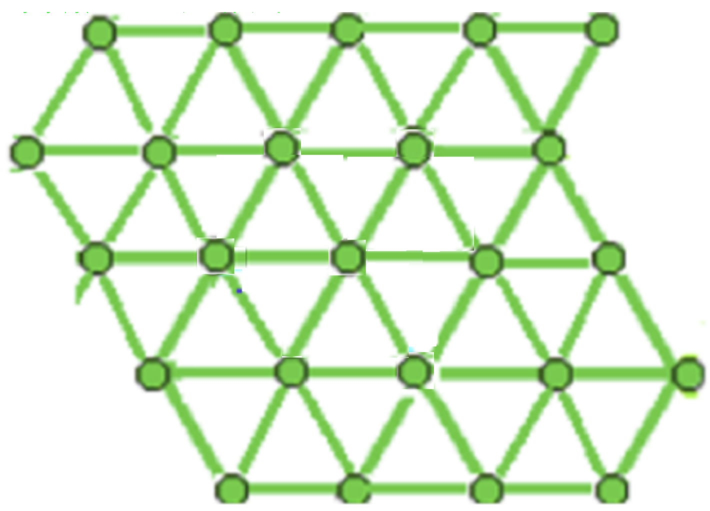


Figure 3.1: Uniform triangular grid.

3.3 Flight segments

This uniform triangular geometry provides the underlying structure for inscribing curved flight path segments, connecting nearby vertices as shown in Figure 3.2. The geometric structure of a triangular grid supports two soft 60 degree turns, and one continued straight path. There are named here as *left*, *straight*, and *right*. Hence, the turning opportunities come in trios of directional options.

One such trio of blue flight segments is shown in Figure 3.3. An aircraft flying into a vertex from a direction near the bottom of the Figure has 3 possible next flight segments, branching to the: left, straight, right toward the top of the Figure. Aircraft flying these linked together path fragments only turn between vertices, never at the vertices themselves (unlike the sharp turns implied in aeronautical charts like in Figure 1.4).

Since the a flight segment branches enter the target vertices at the same angle

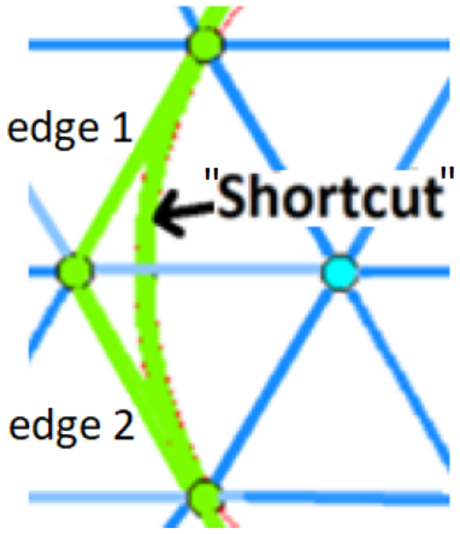


Figure 3.2: Arc connecting two nearby vertices, bypassing and skirting edge 1 and edge 2. Although it is labeled here and looks like a ”shortcut”, it’s actually the flyable physically realizable flight path.

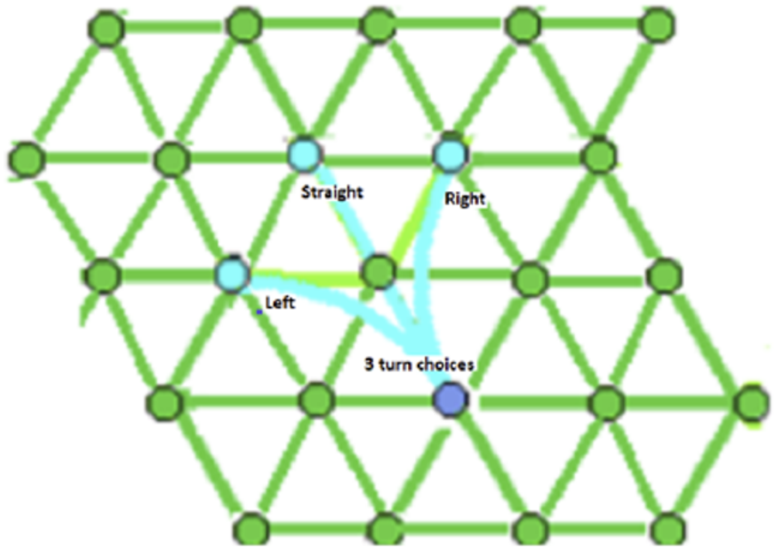


Figure 3.3: Uniform triangular grid with 1 left-straight-right flight path fragment in blue

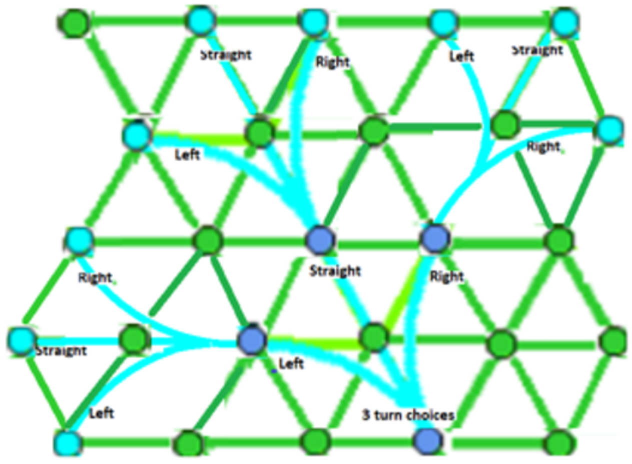


Figure 3.4: Four flight-path segment trios linked together, inscribed on the uniform triangular grid

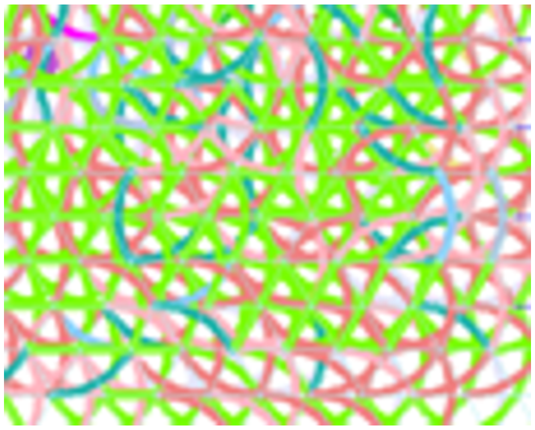


Figure 3.5: Four flight-path segments linked together, inscribed on the uniform triangular grid

and tangent to as the corresponding triangular edge, as well as the beginning of the following flight segment trio, successive flight segment fragments can be seamlessly linked together to construct longer flight paths as shown in Figure 3.4. There is no turning between flight segments, only within the segments.

Figure 3.4 only shows 4 trios of flight path fragments. However, these 4 trios are suggestive the large number of total possible path fragments or segments which fully populate the triangular grid. Figure 3.5 shows the triangular grid fully populated with densely overlaid flight path segments heading in 6 possible different directions in accordance with the underlying triangular/hexagonal geometry. The visual effect in the Figure is saturation by the many possible flight paths, and hence rich optionality in the choice of flight paths.

3.4 Runway final approach

The triangular uniformity and regularity of this triangular approach to automated flight path generation greatly simplify the construction of flight paths. As discussed below, this uniformity also simplifies the search for optimal flight paths too. It's possible as well as mathematically elegant to apply uniform abstractions to a featureless air mass where one air molecule is a good as another.

However, ground features seldom conform to such idealized mathematical regularity. In particular, existing airport runways are unlikely to occupy just the perfect location and angle to mate perfectly with any uniform geometry. Hence, in this thesis project, aircraft are delivered to a waypoint near the beginning of a final approach, instead of the runway threshold itself. The details of the last few miles of final approach flight are well understood, left to the reader, and beyond the scope of this thesis project.

3.5 Path construction from uniform flight segment units

A flight path segment is a fragment of continuous 2D flight path extending from the aircraft's current location to its runway landing location. Figure 3.4 shows only 12 such flight segments. The Figure also shows flight segments concatenated together forming longer 2-segment flight path fragments. This concatenation is at the heart of this *constructionist* satisfiability approach described in this thesis. All flight paths are *constructed* from smaller path segment units. Note that when these flight segments are further annotated with altitude and time, the results are 4DT flight paths, discussed in more detail in Chapter 4.

The underlying strategy of this discreet approach to automating the airspace is to generate large numbers of candidate flight segments with their connectivity affinities that can be smoothly handed off to a satisfiability solver. The solver solves for the optimal flight path (or set of flight paths) among many candidate flight paths.

To accomplish this hand-off to the solver requires uniform units for the combinatorics. This means generating a large number of uniform flight path segments that can be combined through concatenation by adjacent segments, and constructed into large number of full aircraft-to-runway candidate flyable flight paths. It's the perfect congruent uniformity of the flight path segment fragments than enables satisfiability-driven airspace solutions.

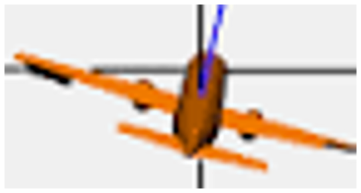


Figure 3.6: Aircraft turn by banking their wings, typically limited to a 20 degree angle

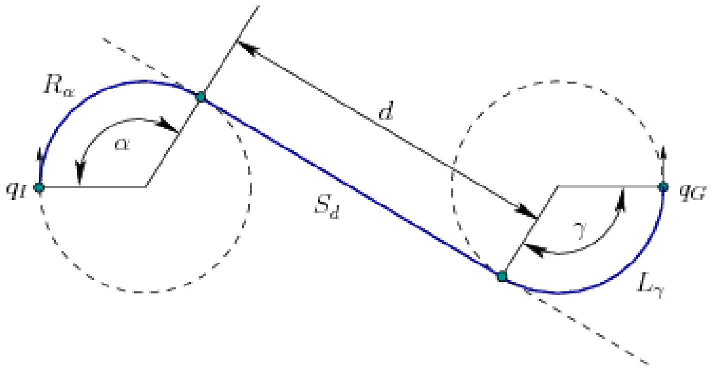


Figure 3.7: A Dubins path is the shortest 2D curve connecting two points in the two-dimensional x,y plane constrained by the curvature of the path. The curves are circular thus conforming to aircraft turn geometry. A Dubins flight trajectory is a concatenation of 3 segments as illustrated here. [65]

3.6 Uniform flight segments analogous to 'pixels'

These flight segments are analogous to pixels in computer graphics. Like pixels, these flight segments are standardized, uniform, and can be combined into a 'larger picture'. Graphics pixels may differ in color, yet can be combined together into many arbitrary visualizations on a screen. Likewise, these congruent 2D flight segments differ in location, altitude, and time, yet can be combined together into many arbitrary flight paths in a computationally modeled airspace.

3.7 Turning and Dubins Paths

The curved segments in this discussion are modeled after the actual geometry of turning by aircraft flying in the airspace, as discussed in this section on Dubins paths.

For fixed-wing aerial vehicles – as distinguished from helicopters and other types of rotorcraft – 3D flight paths are constrained by flight physics (and airline flight policy) to limited climb and descent rates as well as limited turn angle. In the horizontal x,y dimensions of 2D paths, turn limitations are determined by maximum bank angle, illustrated in Figure 3.6.

In principle, an aircraft is limited in changing its flight direction in x,y 2D airspace by its bank angle and corresponding curvature. When an aircraft banks its wings, to a first approximation, its 2D path scribes a geometric circle. This is a Dubins path as illustrated in Figure 3.7. [33] Although most airliner airframes can safely bank at a much steeper angles, for the comfort of their passengers, most airline policies typically limit their pilots to flying bank angles of 20 degrees or less. The informal goal is: don't spill the coffee.

For any particular bank angle, a fixed-wing aircraft scribes a circular 2D path of a corresponding radius. Hence, a flight path though an air mass can be described as a concatenation of straight and circular path segments of some specified radius [97]. Furthermore, the minimum distance path is composed of these segments. Hence, Dubins paths are a useful way to abstract and represent flight paths at the granularity needed for path analysis, namely, on the order of nautical miles.

3.8 Equation for a banked aircraft turn

The radius of a turn varies with the bank angle and the square of the speed of the aircraft. Since curvature is the inverse of turn radius, curvature also varies with bank angle and the speed of the aircraft. The standard formula [93] for turn radius as a function of bank angle used herein is equation 3.1 as follows:

$$
\begin{gathered}
R = \frac{V^2}{\tan\theta} \\
\text{where } R = \text{turning radius} \\
V = \text{true airspeed} \\
\theta = \text{bank angle}
\end{gathered}
\tag{3.1}
$$

3.9 Two representations of flight path segments

Flight paths and their flight segment component parts are represented in two different manners in simulations here. Each representation has its distinct use and visualization. These two representations have closely related geometries, but function in different ways in the model:

- Diagrammatic: composed of pure edges of the triangular grid. Not flyable per se, but simpler for some calculations and searches. Contains the underlying structure for the flyable flight segments.

- Smooth, flyable visualization: composed of curved arc or straight flight segments, representing the routes that aircraft can fly, but rooted conceptually and geometrically in the simpler diagrammatic pure triangular representation.

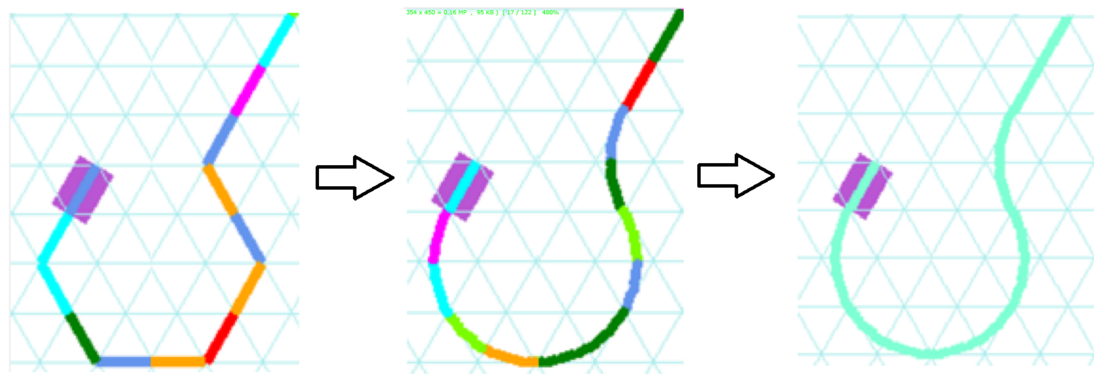


Figure 3.8: Three representations of the same flight path. Left: diagrammatic form composed of edges, multi-color to highlight composite structure. Middle: smooth flyable form composed of flight segments, multi-color to highlight composite structure. Right: flyable form composed of flight segments, uni-color to highlight continuity

The two representations can be easily mapped between each other if well behaved. While every set of contiguous flight segments has a corresponding set of contiguous triangular edges, not every set of contiguous edges has a meaningful set of flight segment counterparts.

The flyable representation is composed of curves when encoding aircraft turns. The curved segments link together, and as such are a shortcut between two adjacent turning edges. A curved segment must be completed before a new turn can begin. Hence, two successive turning vertices may not occur. At most every other vertex may turn.

Figure 3.8 illustrates the two types of flight path representations described above for the same flight path. The version at left is composed of triangle edges, while the version at right is composed of flyable flight path segments. This Figure illustrates smooth flight paths constructed from underlying edge-based structure.

3.10 Flight path curves and safe aircraft separation

So far we have discussed the radius of an aircraft turn as limited by flyability at a maximum of 20 degree bank angle. There are actually two important radius constraints:

- The curve must be flyable at all anticipated aircraft speeds
- The nearby flight path segments must be safely separated: at least 5 nm apart

In this section we address the safe separation of coincident nearby aircraft. Ideally, the geometry would itself enforce safe separation of aircraft flying on different flight segments, even if nearby. The goal here is that as long as multiple aircraft do not fly co-temporally on the same flight path segment, then the worst-case geometry distances will always be greater or equal to 5 nm (nautical miles). Note that required aircraft separation by the FAA is 3 or 5 nm depending on altitude. So, 5 nm separation is always a universal minimum safe distance apart aloft. (As discussed further below in Section 4.2 on synchronized landings there can be an exception near the final approach where 3 nm separation is adequate and conforms to FAA requirements.)

The overall goal here is to ensure flyability and safe separation in the air while automatically solving for the optimal set of flight paths. This is accomplished by constructing all flight paths from interchangeable parts designed for uniform combinatorics and handed off to a satisfiability solver. This is the rationale here for such fastidious standardization of the flight path segment component parts.

These shortcut flyable path segments are designed to accommodate aircraft flying at a full range of air speeds, from greater than 500 knots down to much slower

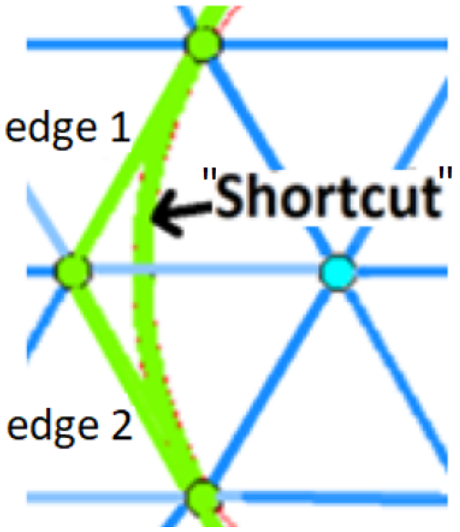


Figure 3.9: Flight path appearing to be “shortcut”, skirting edge 1 and edge 2 and the intermediate vertex, where the endpoints are vertices. Rather than a “shortcut”, the path is actually the correct realizable flyable path between the two endpoint vertices.

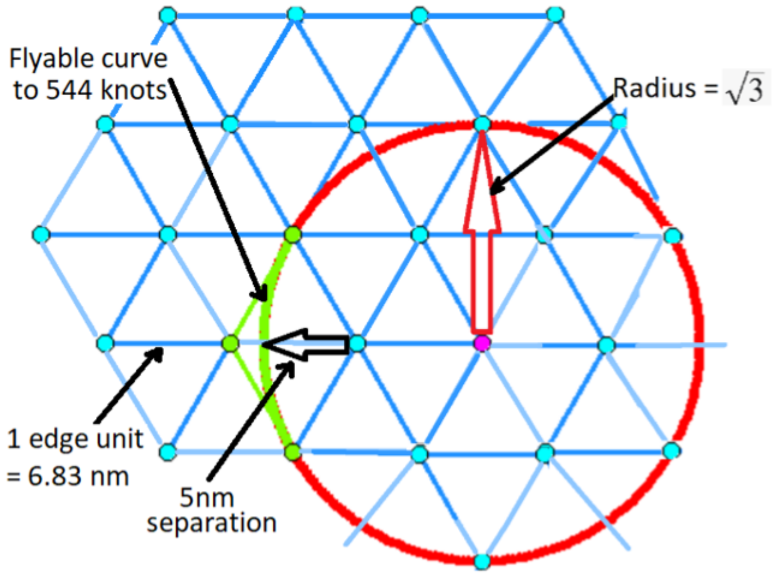


Figure 3.10: Flight path shortcut showing the fuller geometry

runway approach speeds. One size fits all. The squared term in the aircraft banking radius formula in Section 3.8 means that the faster the aircraft speed, the larger the required radius of the curve for turning. Therefore, the minimum required radius is calculated for the worst case top speed of aircraft. Hence, this becomes the universal standard curvature used across the entire airspace for all aircraft, regardless of whether the radius is unnecessarily large for aircraft flying at lower speeds.

Figure 3.10 shows a green arc flight segment embedded in the full red circle of radius $\sqrt{3}$ edge units as illustrated by the large red arrow. The black arrow shows a 5 nm minimum separation between the curved flight segment and the nearest vertex which is not part of this nominal straight-line edge flight path.

This black arrow has a length of 5 nm. Edge units are therefore 6.83 nm in length. A simple geometric calculation confirms this value:

$$\frac{5 \text{ nm}}{\sqrt{3}-1} = \frac{5 \text{ nm}}{0.71} = 6.83 \text{ nm} \qquad (3.2)$$

However, would this 6.83 nm shortcut flight segment arc be flyable at all anticipated aircraft speeds? The following calculation applying the formula 3.1 in Section 3.8 for the radius of banked turn confirms that these properly separated flight segment arcs are indeed flyable with a substantial safety margin:

First consider a 6.83 nm edge, and calculate the size of curve radii:

6.83 nm edge = 11.83 nm radius

= 21950 m radius

Next consider an aircraft flying at 544 knots

544 knots = 280 m/s

Calculate the radius size for turning at a 20 degree bank angle at 544 knots:

$$\begin{aligned} R[m] &= \frac{V^2}{9.81 m/s^2 \tan\theta} \\ &= \frac{280 m/s^2}{9.81 m/s^2 \tan 20 deg} \\ &= \frac{280 m/s^2}{9.81 * 0.36397} \end{aligned}$$

= 21959 m radius

Therefore a 6.83 edge has an adequate radius to fly an aircraft at a speedy 544 knots with no more than a 20 degree banking turn in these flight segments

3.11 Chapter conclusion

For this airspace modeling project, the triangular edge length can therefore be universally set to the constant of 6.83 nm. Using this convention, we can now proceed to discussing the proposed airspace geometric structure for automatically guiding in aircraft from their current positions aloft, descending to a specific runway approach in full 4DT geometry. All edges will be 6.83 nm long, and all flight segments will scribe a shortcut arc between nearby vertices 2 edges apart as illustrated in Figures 3.9 and 3.10. This insures flyability as well as proper safe separation of aircraft aloft as long as the discipline of unique coincident use of edges in enforced. Thus optimum flight paths can be generated with minimum computation.

Chapter 4

Geometry - Part 2: 4DT Paths

4.1 Introduction

Chapter 3 addresses 2-dimensional (2D) xy geographic flight paths. In this chapter, the additional two dimensions of z altitude and t time are added and discussed to complete the 4 dimensions of navigation, addressing full 4DT flight paths.

4.2 Runway sequencing and aircraft separation on landing

Safe separation of aircraft aloft is discussed above in Section 3.10. This separation is accomplished by careful choice of the triangular edge length of 6.83 nm such that nearby flight paths will always remain at least 5 nm apart geographically. This intrinsically enforces separation aloft passively through pure geometry. No appeal to employing additional, active algorithmic mechanisms is needed, thus saving computational cycles which would otherwise be dedicated to active enforcement of separation minima.

A goal for this thesis project is also to enforce safe runway sequencing by enforcing safe separation upon landing through passive geometric means as well. This requires some additional structure to fully accomplish safe landing separation and proper runway sequencing though.

4.3 Sparse airspace – crowded runways

Even when pilots tell their passengers that the airspace is so crowded that the associated congestion is causing delays, a quick glance out the windows suggests otherwise. Compared to the vast size of the airspace, there are relatively few aircraft "up there." The density of aircraft aloft is sparse – except near runways.

The most precious spacial resource is at runway thresholds. Airports limit the number of "operations" or allowed landings per unit time, typically to about 60 "ops" per hour, or about one landing per minute. Accordingly, in this project, landings are limited to once per minute, and furthermore simplified, regimented, and synchronized to landings on the precise minute mark. This rigid on-minute landing discipline enables safe separation of aircraft at runway thresholds. This synchronized sequenced landing discipline is accomplished through special airspace structure discussed below.

4.4 Pre-calculated airspace structure

For all descending aircraft flying towards some set of runways, there is always some prima facie nominal current shortest path to the closest runway, for the current set of flight airspace conditions (e.g. weather), independent of other traffic if any. These shortest paths are calculated efficiently once at the beginning of each planning cycle, for all aircraft in the local airspace.

Figure 4.1 shows a set of pre-calculated flight paths for a subset of initial flight segments descending to a single airport runway, These are “pure” descent paths calculated as if the airspace were empty, i.e. as if there are no other aircraft in the airspace. These nominal flight paths are calculated by ascending backwards up from each runway, flooding the space to all path segments. Many redundant flight paths are explored, but only the shortest path for each flight segment is preserved. This airspace structure is used to synchronize descents so the all aircraft land on some precise minute mark, as discussed below.

Pre-calculated flight paths for many positions and curves are not shown in Figure 4.1, as the visualization would be a smear of color, almost solid with lines and curves, and therefore poorly illustrative. Figure 4.1 is suggestive of the pre-calculated airspace structure, while not showing needless detail.

Regardless of the incomplete, sparse visualization, suffice it to say that all shortest flight paths are indeed pre-calculated from all possible aircraft positions and headings, to all available runways. At the beginning of each planning cycle, hypothetical shortest flight paths from all possible aircraft locations to runways are pre-calculated completely in advance, as if there’s no traffic. The enables more efficient rapid exploration of alternative flight paths during the planning cycle when direct flight paths are not possible due to the presence of other traffic. No matter how off course an aircraft needs to fly to avoid other traffic, the shortest path from where it has to go to a runway has been pre-calculated. This is very important for sequencing aircraft into runways on

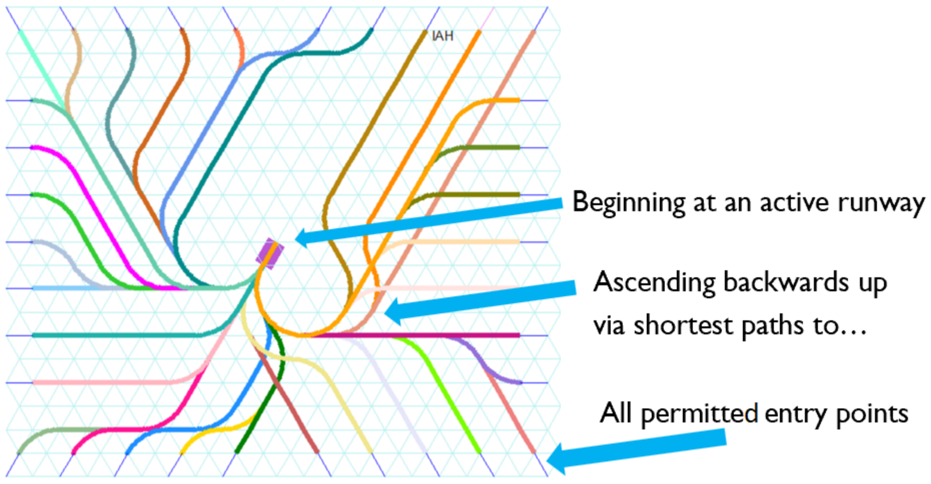

IAH
Beginning at an active runway
Ascending backwards up via shortest paths to…
All permitted entry points

Figure 4.1: Shortest path airspace pre-calculated airspace structure

the minute mark.

4.5 Temporal pre-calculated airspace structure

Aircraft must be safely separated aloft as well is upon landing. Safe separation aloft is defined in terms of inter-aircraft distance. Safe separation on landing can also be defined in terms if aircraft time spacing. This is because requiring aircraft to land on a precise minute clock tic, implicitly arranges arriving aircraft to be separated by a safe distance. Therefore, the separation distance requirement can be translated into safe temporal separation. This is convenient shift of constraints.

In order for aircraft to arrive on the precise integer minute mark, their minute-by-minute time discipline must be maintained not only on landing, but also in the several minutes prior to landing. This is analogous to surfers riding a wave toward the beach as illustrated in Figure 4.2. In this aircraft case, the waves are separated by exactly

Figure 4.2: Temporal spacing of arriving aircraft is analogous to surfers riding a wave

one minute all the way to the runway threshold (or more precisely the point of the final approach as discussed in Section 3.4).

Once the shortest flight paths from all path segments to the closest runway are calculated as shown in Figure 4.1, the next step is to annotate each aircraft's future projected locations with precise minute marks as shown in Figure 4.3.

For each aircraft and its closest runway, according to its current position, direction, speed, and altitude, reasonable flight parameters, the minute-by-minute time discipline is calculated. The projected location of the aircraft at each on-minute mark is shown in Figure 4.3. If the aircraft adheres to the temporal discipline of flying past each minute mark waypoint on the minute as calculated and as shown, the aircraft will touch down in precisely 18 minutes on the minute.

This is only a plan though, calculated for the succeeding 18 minutes, but only definitely flown for the ensuing 1 minute. After 1 minute, the plan may be altered in the

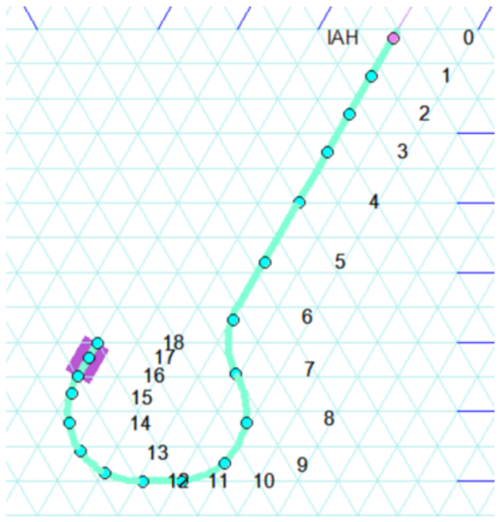


Figure 4.3: Aircraft planned to be partway along flight path segment at each precise minute mark, i.e. an integer number of minutes projected until landing.

next planning cycle, due to changing airspace conditions if any. Although this plan may be altered in subsequent planning cycles, this plan has a reasonable chance of remaining stable, guiding the aircraft on a temporally optimal path to the runway. If flight path or timing changes are later required, those calculations can be made at that succeeding time. But for that planning cycle, this is the plan that is flown for the next minute (as noted in Subsection 2.3.4).

A flight path is a concatenation of multiple path segments as shown in Figures 3.4 and 3.8. The location of an aircraft at any designated minute mark is somewhere on the overall flight path, somewhere along a particular path segment as illustrated in Figure 4.4. The location of a descending aircraft is easily calculated to a specific path segment, and its precise position within that path segment.

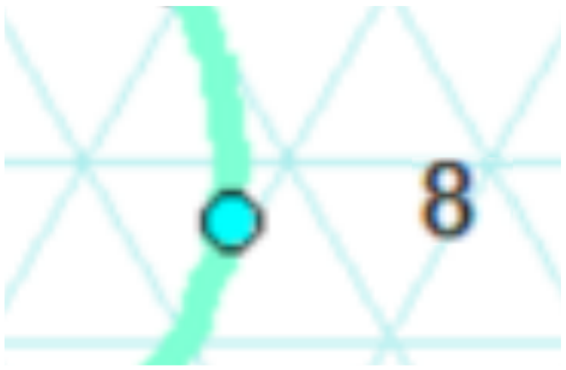


Figure 4.4: Aircraft planned to be partway along flight path segment on a minute mark

4.6 Descent profiles

The pre-calculated xy-horizontal structure discussed in Section 4.5 also contains implicit z-vertical structure as well. Figure 4.1 shows a sample of pre-calculated shortest paths to a runway. These shortest paths, shown and not shown, are the shortest possible paths to the closest runway for any given aircraft and heading. In order for the aircraft to land at zero altitude, it must descend some minimal vertical distance during each time period of its descent.

Figure 4.5 shows a set of typical altitude descent profiles for an ensemble of descending aircraft in the local airspace. Each black curve shows the planned descent altitude and time profiles for each aircraft. In this example, there is only one available runway for landing. Note that all aircraft are planned to land on a unique integer minute. This indicates that sequencing is planned for one or fewer landings (operations or 'ops') per minute.

Also note that in Figure 4.5, almost all of these altitude profiles are descending monotonically (at least in this planning cycle's plan). However, there are 3 horizontal flat portions of 3 of the flights. These are the signatures of these 3 aircraft whose paths

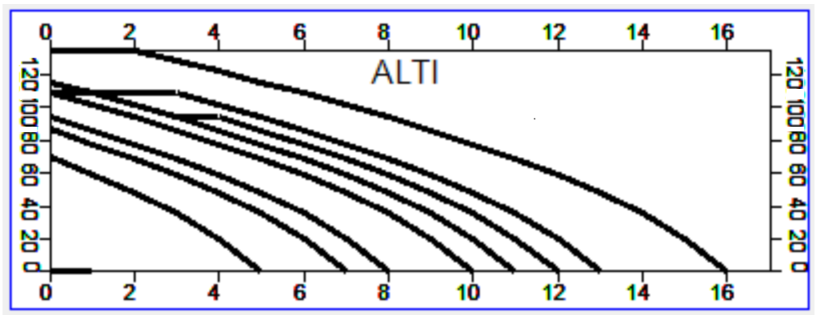


Figure 4.5: Aircraft altitude profiles, descending out of 12000 ft.

were automatically altered away from their shortest path to landing due to separation and sequencing issues, automatically favoring a slightly longer, slightly shallower descent in places, but safe path to the runway.

As will be discussed in the upcoming Chapters 5 and 6 on Combinatorics and Satisfiability, vast numbers of candidate flights paths are evaluated for the ensemble of aircraft. This process is where flight plans are calculated which conform to optimal proper separation and sequencing.

4.7 Disruptions

One of the benefits of the automated aviation system proposed in this thesis is its adaptability and robustness in response to continually changing airspace conditions, as discussed in Section 2.3. These changing conditions include weather, runway availability and reversals, missed approaches, as well as other air traffic. The current best plan for the current minute may need to be revised as soon as the next minute arrives, as noted in Subsection 2.3.4.

In general, any event that requires a revision of flight plans can be classified under the universal rubric of “disruptions”. Accordingly, the system described here responds in a general way to disruptions, regardless of the specific type of disruption. Nevertheless, some particular disruptions are addressed individually below.

4.8 Weather and Closed Airspace

The triangular grid workspace represents a portion of the entire airspace, structured with its pixel-like flight segments. All triangular vertices, and their corresponding flight segments, have a state of either ’on’ or ’off’. The default state is ’on’. If for any reason, some region of the airspace needs to treated as closed airspace, a status change is performed, switching some number of triangular vertices to an ’off’ state. Similarly, when re-opened, the state of the triangular vertices is switched back to ’on’. Figure 4.6 shows how a storm is represented by blocked (red) vertices.

In general, any closed airspace is treated this way, by switching off the state of the corresponding vertices. One example of closed airspace is the region in or near a storm or convective weather. Figure 4.7 as well as the right-hand panel of Figure 2.2 illustrate such storms, with flight paths avoiding that region of airspace by detouring around them.

Also, since convective weather may be moving in an unpredictable direction, a “buffer” of grid nodes immediately adjacent to the convective weather is also maintained as as slightly wider path of blocked vertices, as illustrated in Figure 4.6. This ensures that aircraft remain far enough away from a storm so that the storm doesn’t overtake the aircraft in the next planning cycle (or between planning cycles).

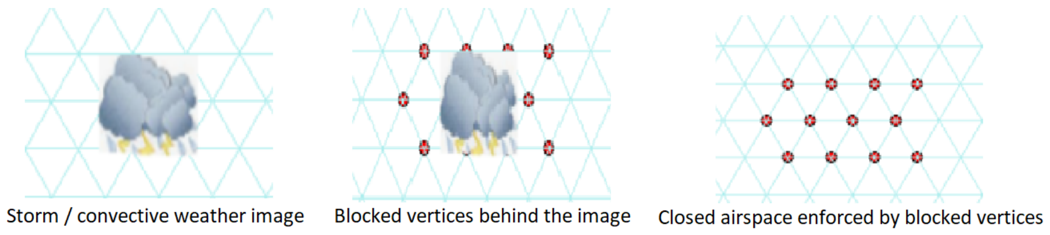

Storm / convective weather image
Blocked vertices behind the image
Closed airspace enforced by blocked vertices

Figure 4.6: Convective weather is represented as closed airspace, via switching some number of vertices to an ’off’ state (shown in red), repelling exploration of possible flight paths through those vertices. Note that the convective weather drawn as a storm image visually obscures some of the closed red vertices here, but functionally the closed red vertices are still present and operational.

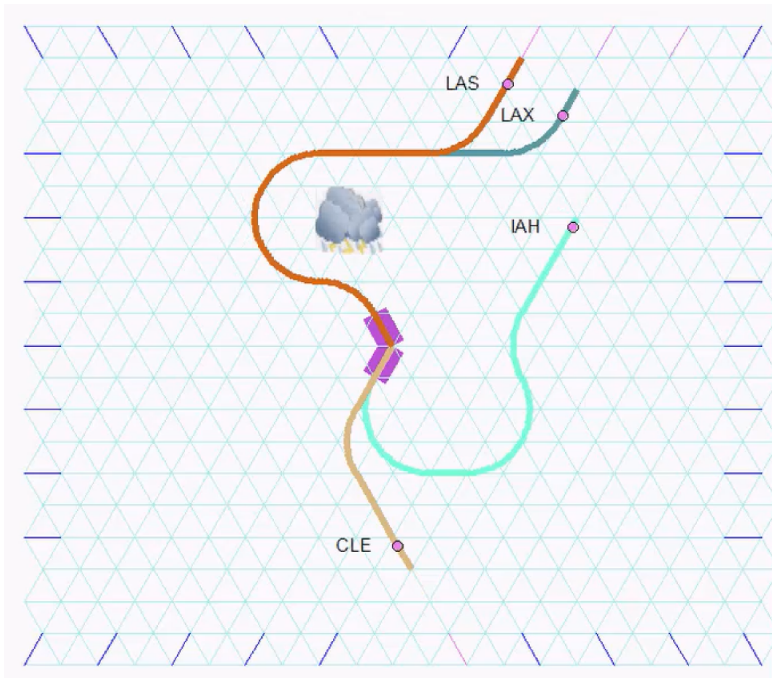


Figure 4.7: Convective weather represented as closed airspace.

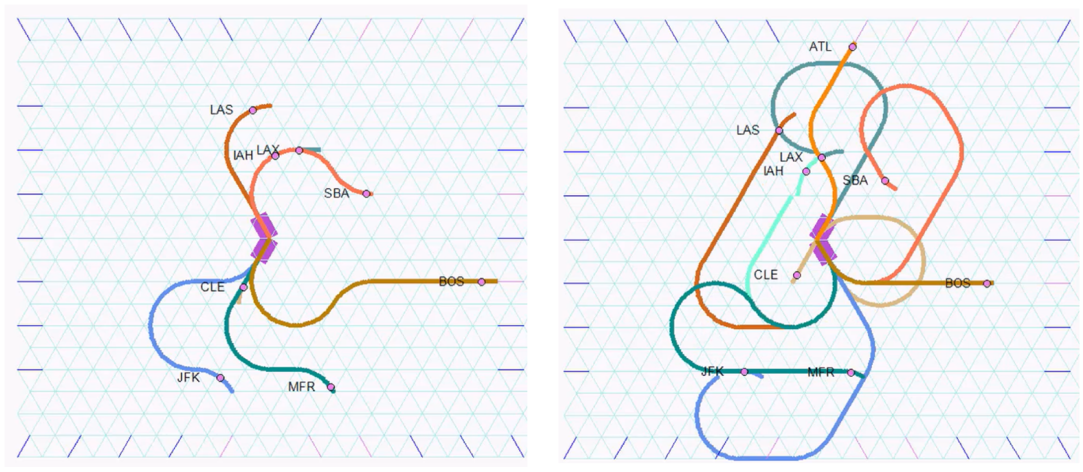


Figure 4.8: Two panels showing before and after a runway reversal for 2 runways drawn as purple rectangles. New flight plans are immediately calculated for the right panel. However, the aircraft are now typically farther away from the newly reversed runways.

4.9 Runway reversal

In order to minimize aircraft ground speed on landing, the runways are typically assigned such that the aircraft will be landing into the wind if possible. However, when the wind shifts direction, it is typically best practice to reverse the assigned directions of the runways.

This abrupt directional change can disrupt all or most of the flight paths of arriving aircraft. Smooth response to runway reversals is a difficult problem for Air Traffic Control (ATC), and typically requires considerable manual ATC intervention. Indeed, the human scramble to reroute arriving aircraft manually takes time, and can result in unused landing slots.

Newly reversed runways are typically used inefficiently for a period of time just after the reversal. This is an artifact of geography. This is because aircraft are now typically farther away from the newly reversed runways. The automated free-flight

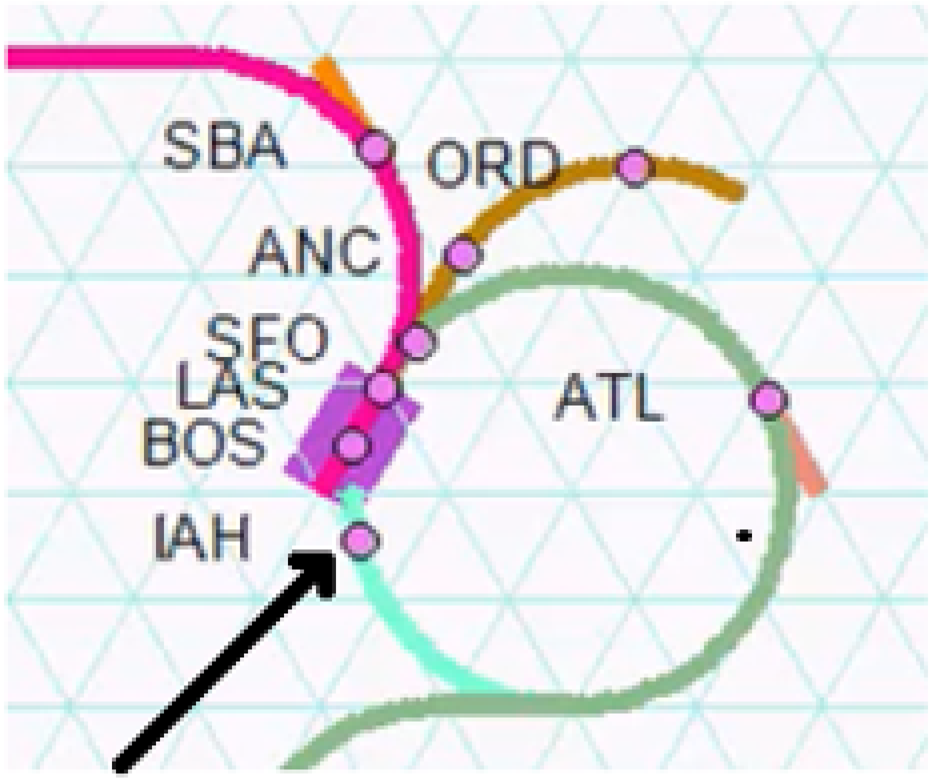


Figure 4.9: Missed approach by aircraft arriving from IAH (black arrow). In the sequence of landing aircraft, IAH was in the front of the queue, but failed to land. So, its flight path is seamlessly re-calculated in the next planning cycle just like any other aircraft. Its new flight plan is to circle around and fit into the landing queue with other aircraft on descent.

system proposed in this thesis has promise in utilizing valuable runway landing slots more efficiently immediately after runway reversals. But, this is speculation, and will require modeling and testing to validate, which is outside the scope of this thesis.

The automated system proposed in this thesis responds seamlessly to runway reversal disruptions. When there is a runway reversal, the system re-calculates the new revised flight paths in the next planning cycle, as illustrated in Figure 4.8.

4.10 Missed Approach

Another disruption is a missed approach. This is an unexpected change in landing plans when an aircraft abandons its runway approach. Instead of touching down, the aircraft continues aloft. In this case, the aircraft is poised to land, but at the last moment, the pilot aborts the landing in favor of continuing its flight from its already low altitude, to land a few minutes later instead. There are many possible reasons for this, including unsafe runway conditions on the ground, wind issues, pilot error, etc. In any case, the landing is skipped, and the aircraft needs to land at some future time.

When the aircraft does finally land, it may land on the same runway, or a different runway. Once the aircraft has missed its planned landing slot, it is treated identically to any other aircraft on its way to landing in the local airspace. It is neither penalized, nor given special status priority. In fact, it is not given any special consideration. This is an example of the generality of the proposed system.

Figure 4.9 illustrates a missed approach, or actually immediately after the missed approach, after the next planning cycle. In this Figure, the system has already calculated a new flight path for the flight from IAS (see large black arrow), and placed it in the de facto landing queue. Of course, some new disruption may occur a minute later, but this Figure shows the immediate seamless way of responding to a missed approach.

4.11 Chapter conclusion

By adding the zt-dimensions to the xy-geographical discussion in Chapter 3, full 4DT flight paths are represented in this chapter. The next task is choosing the optimal safe set of flight paths out of the vast numbers of possible candidate flight paths. This winnowing can be done systematically using concepts drawn from the fields of combinatorics and satisfiability, discussed in the two following Chapters 5 and 6.

Chapter 5

Combinatorics

5.1 Introduction

The goal of this thesis is to propose a novel way to automate the construction of safe optimal flight paths for an ensemble of arriving aircraft descending into available runways of an airport. When rigid STARs navigational requirements are relaxed in favor of an automated version of free-flight, the number of ways for an ensemble of aircraft to fly their terminal routes becomes truly vast. There are even a large number of safely separated ways for aircraft to fly their terminal routes.

The previous two chapters 3 and 4 define the underlying the geometric structure of this formalized airspace. This geometric structure enables a systematic enumeration of vast numbers of possible *ways* to descend and land an ensemble of aircraft. These ways are constructed in a similar manner from the same repertoire of similar Lego-like flight segments, discussed in Section 3.3

The theme of this chapter are the possible *ways* of flying the airspace, in this case aircraft flying their terminal routes. The underlying geometry is discussed in chapters 3 and 4 that enables enumerating and placing these possible *ways* logically in parallel. In this fashion, these *ways* can be evaluated and treated in a general, universal manner.

5.2 Multiple Flight Paths

5.2.1 Hypothetical individual solution approach

If there were only one aircraft in the local airspace, the lowest cost way to manage the airspace would be for this solo aircraft to fly its lowest cost 4DT path to landing. This aircraft would simply follow the shortest, most efficient 3D path to the nearest runway.

However, typically, there are multiple aircraft in the local airspace. Furthermore, these aircraft will have competing demands on resources, namely airspace and landing times. An aircraft‘s first choice may conflict with another aircraft‘s first choice.

In an individual aircraft approach to “solving” the airspace, each aircraft would need to entertain alternative 4DT paths or *options* in case its first choice were not feasible. Each 4DT path has different costs and corresponding preferences. For each planning cycle, for each aircraft, a set of candidate paths would be generated by the system, from which exactly one 4DT path would chosen for each aircraft by the global planning system.

In this individual approach, each aircraft’s optimal flight path is evaluated one by one, one after the other. Clearly, the first aircraft would have an advantage, because it would be endowed with the greatest *optionality*. Succeeding aircraft would have ever decreasing optionality, and thus, on average, decreased optimality.

5.2.2 Ensemble group solution approach

In principle, the individual solution approach discussed above will tend to miss optimal solutions for the full ensemble of aircraft, where the best solution for the ensemble group requires a sub-optimal solution for the initial or other aircraft evaluated. Indeed the optimal solution may not be the individual optimal solution for any of the aircraft in the group. There can be a optimality conflict of interest between individual aircraft's optimal routes and the optimal set of routes for constellation group of aircraft taken as a whole.

Hence in principle, the optimal group solution must consider the vast number of flight-path possibilities for the group of aircraft, with no biases toward any individual aircraft in the group or ensemble of aircraft. As discussed in more detail below, this may not be fully computationally practical at present, but it's important to keep in mind as an objective and methodology.

5.3 Combinatorics

One of the goals of this thesis is to systematize the search for safe optimal sets of flight paths for aircraft descending to available runways, calculated at each planning cycle. Automation requires systematizing. Systematizing requires a repertoire of one-size-fits-all, universal, pixel-like, Lego-like flight-path construction units. Here, it is the path-segment, as discussed in Section 3.3, that serves as the universal unit of construction of full flight paths.

These geographical flight segments are further annotated by z-altitude and t-time. For the discussion and examples here, only the xy-geographical flight segments are shown in the numbering of the geometric edges. Additional z-altitude and t-time dimensions can be added similarly.

5.4 Connectivity matrix

In order to construct full flight paths from individual path segments, path segments must be stitched together, one after the other, into a contiguous sequence of flight segment units. Geometrically, a flight segment links two edges together forming either a linear pair of edges, or two possible soft left or right 60 degree turns. As shown in Figure 3.3, each edge participates in a trio of flight segments corresponding to navigating relative directions *left*, *straight*, *right*.

Path segments are allowed to be linked together if and only if the 2nd ending edge of one flight segment is identical to the 1st beginning edge of another flight segment as illustrated in Figure 3.4. To support this requirement, each edge is assigned a unique (arbitrary) integer number. Hence two flight segments can be joined together by matching identical flight segment numbers.

Table 5.1 shows an adjacency matrix where edge number 10 is connected to a trio of edges 21, 22, 23, corresponding to the relative navigational directions left, straight, right emanating from edge 10.

Due to the implicit left-straight-right trio structure of flight segments, the

From	To
10	21
10	22
10	23

Table 5.1: Simple three column adjacency matrix, encoding three flight segments made up from the pair edges 10 and 21, edges 10 and 22, edges 10 and 23

From	To	To	To
10	21	22	23

Table 5.2: Compact 4-column single row version of three rows of the previous 2-column adjacency matrix

simple from-to adjacency matrix in Table 5.1 can be compacted to the four column matrix shown in Table 5.2.

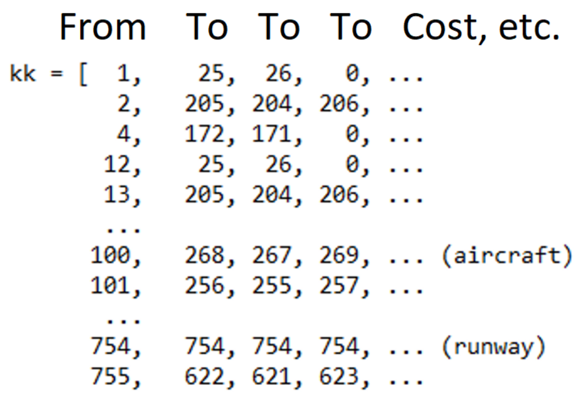
With this four column from-to-to-to style of matrix, the ingredients for connected flight paths constructed from individual flight segments can be efficiently encoded as exemplified in Table 5.3

Using this style of matrix, multiple trios of flight segments can be encoded as follows. Since edge 23 is an element of one of the "To" columns of row one, and also an element of the "From" column of row two, the existence of three connected pairs of line segments can be inferred, namely, 10-23 connected to 23-31, 23-32, 23-33.

Putting all this together, the exhaustive matrix encoding the full set of possible flight segments and their inferable connectivity can be constructed. Figure 5.1 shows an excerpt of matrix enumerating all possible Lego-like flight segments, ultimately encoding

From	To	To	To
10	21	22	23
23	31	32	33

Table 5.3: Matrix encoding two rows of flight path trios, enough to construct some two segment flight paths



```
            From    To   To   To   Cost, etc.
kk = [  1,     25,  26,   0, ...
        2,    205, 204, 206, ...
        4,    172, 171,   0, ...
       12,     25,  26,   0, ...
       13,    205, 204, 206, ...
       ...
      100,    268, 267, 269, ... (aircraft)
      101,    256, 255, 257, ...
       ...
      754,    754, 754, 754, ... (runway)
      755,    622, 621, 623, ...
```

Figure 5.1: Matrix enumerating all possible Lego-like flight segments, ultimately encoding the combinatorics of the vast number of ways to construct flight paths. In this representation of connectivity as left-straight-right trios of next possible flight segments, column 1 contains a flight segment id, and columns 2,3,4 contain the ids of allowed following flight segment ids. Hence segment 1 may be connected to any of the segments 25,26,0. Matrices expressing connectivity like this tend to range from 1,000s of rows to 10,000s of rows depending on the size of the airspace being modeled and solved. Note in the following chapter on Satisfiability, simple 1-to-1 from-to matrices are used, instead of 1-3 matrices.

the combinatorics of the vast number of ways to construct flight paths for the descending aircraft.

The current locations of the flight segments of aircraft aloft in the airspace are assigned special numbers (100, 101). Similarly, the locations of final flight segments of available runways are also assigned special numbers (754, 755). The only interesting flight paths are those contiguously linking a aircraft with and available runway. Hence, those flight paths will begin with an aircraft flight segment (100, 101) and end with a final approach flight segment (754, 755). Typically, there will be vast numbers of such flight paths. Some combination of those flight paths across all the aircraft currently aloft in the airspace, are the optimal set of safe flight paths.

The final choice for the set of flight paths for the ensemble of aircraft for each planning cycle must also be safely separated aloft. Fortunately, the geometry discussed in Section 3.10 on separation issues aloft simplifies this problem considerably. As long as flight paths do not occupy the same vertex at the same time, the paths are implicitly safely separated.

Although the matrices could also include altitude to theoretically pack even more flight paths in the airspace using altitude separation accomplish this, the examples here are kept simple for explanatory purposes.

5.5 Chapter conclusion

The goal of this thesis is propose a novel way to automate the construction of safe optimal flight paths for a group of aircraft. The goal of this chapter was to propose a formal systematic way to make possible evaluation of large numbers of candidate flight paths in favor of a set of an optimally safely separated set of flight paths to be flown for the next minute.

Enumeration of the set of possible flight paths using the same standardized fundamental unit of construction enables use of general tools to winnow these vast combinatorics towards the goal of a single particular optimal solution. The following Chapter 6 discusses this approach using general (and specialized) solvers.

Chapter 6

Satisfiability

6.1 Introduction

There are many areas of computer science and engineering that require solving constraint satisfaction problems, including aviation. As discussed in this thesis, there are many constraints to safely flying a suite of aircraft descending into available runways while preserving proper sequencing and separation of aircraft.

These constraint problems are so ubiquitous and important that constraint problem solving has become its own specialized field, often called *satisfiability* or “SAT”. In mathematical logic, a boolean algebraic expression is said to be *satisfiable* if the expression evaluates to true under some assignment of values to its boolean variables. For large boolean expressions, it can be difficult to find the right set of boolean values to satisfy the entire expression.

Such boolean expressions, made up of *and*, *or*, *not* operators can be transformed into a stylized conjunction of clauses of disjunctive sub-expressions, called conjunctive normal form, in particular *3-SAT*. Satisfiability is thought to be an example of a NP-complete problem, although Don Knuth has speculated otherwise [63].

Supporting the wide interest and usefulness of Satisfiability, generalized *solvers* have been built, and are widely available. There are also specialized high-level languages designed to express these constraint problems succinctly, in concert with a repertoire of specialized plug-in solvers, targeted at specific genres of constraint problems. One such high-level constraint language, *MiniZinc* [73], is used in this thesis project, and discussed in more detail in Section 6.3 below.

6.2 Airspace automation as a satisfiability problem

Given the problem-solving power and existing resources available in the Satisfiability field, automating aviation is a good match for applying constraint methodology, and satisfiability machinery in particular.

One of the challenges for the airspace automation described in this thesis was to form and set up the overall problem as a well-behaved constraint satisfaction satisfiability problem. This was accomplished through a careful use of specific geometry together with standardized uniform fundamental flight segment units to construct full aircraft-to-runway flight paths. This setup has enabled generating the combinatoric enumeration described in the previous Chapter 5 on Combinatorics, with the formalization of generating the vast number of *ways* to fly the airspace in the form of matrices.

The guiding idea and insight behind this thesis is to translate the airspace automation problem into a *satisfiability* constraint satisfaction problem that can be "fed" into an off-the-shelf satisfiability *solver*. As discussed below, at present this pure generic solver approach is too computationally intensive to act as a turn-key solution for practical problems, but someday this generic approach will likely be feasible. Meanwhile, a custom approach is described further below in Section 6.7.

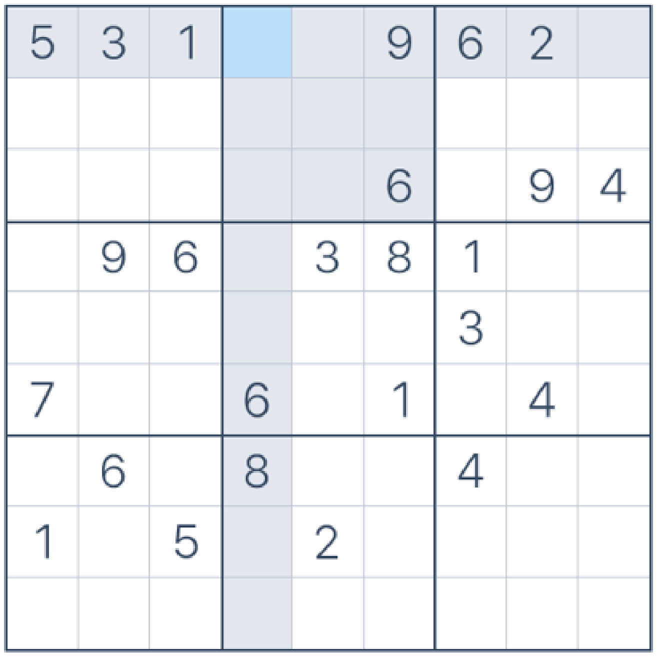
6.3 MiniZinc language - Sudoku example

As discussed above, *MiniZinc* is a high-level language specifically designed to express and solve constraint satisfaction problems. There are many such problems that would take hundreds of lines of code to express in a typical algorithmic language like C++, but can be expressed and solved in just a few lines of MiniZinc code.

As a preface to discussing applying MiniZinc to aviation, the simpler example of solving the popular fill-in-the-digit Sudoku puzzle can help showcase the principles and notation of constraint satisfiability languages, and MiniZinc in particular.

Figure 6.1 shows a typical Sudoku puzzle. It's a 9x9 table of single digits to be filled in, some already filled in at the beginning, the rest to be filled in by the player to solve the puzzle. The constraints are threefold: no duplication of digits in any row, column, or 3-3 sub-table as delineated in the Figure 6.1.

The few lines of code needed to solve the problem in Figure 6.1 are shown in Figure 6.2. Apart from the declaration of variables, output etc., and using special one-off constraints to fill in the initial position values in the 9x9 table, there are only 3 lines expressing the 3 general constraints that are the core of the Sudoku rules. These three core Sudoku constrains enumerated in the previous paragraph are highlighted in



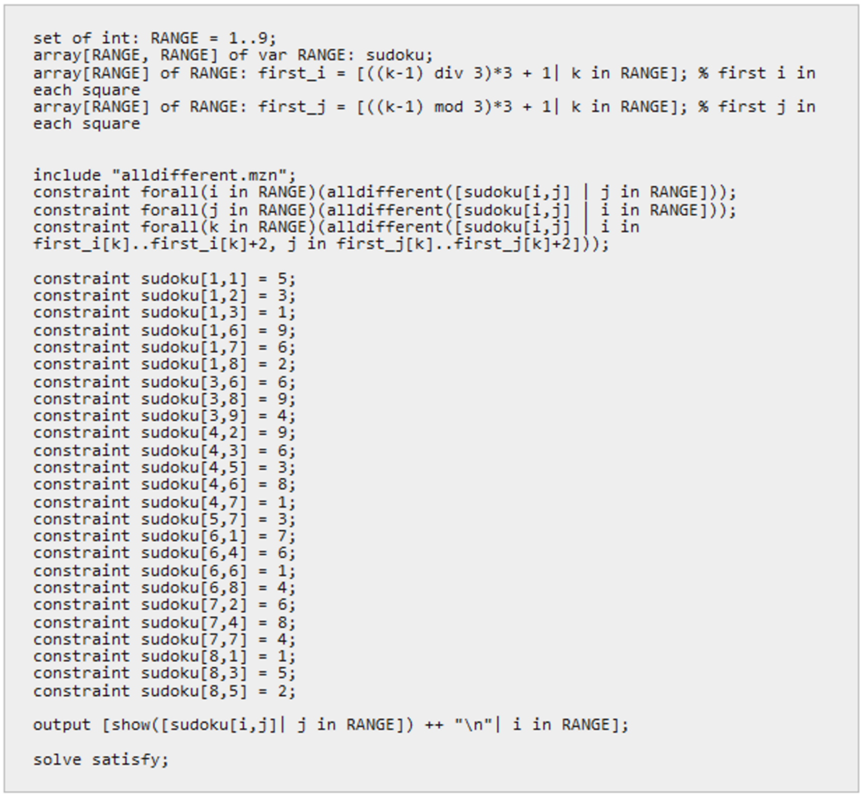
5	3	1			9	6	2	
					6		9	4
	9	6		3	8	1		
						3		
7			6		1		4	
	6		8			4		
1		5		2				

Figure 6.1: Sudoku example.

Figure 6.3. They express each of the three constraints on the placement of the digits in the rules of the Sudoku game.

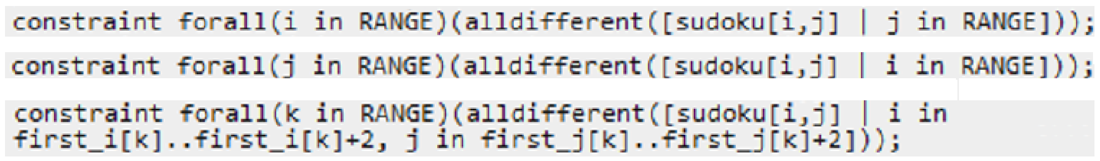
The last line in Figure 6.2, "solve satisfy," sets the solving process in motion. Under the covers, there is a satisfiability *solver* engine searching for some choice of boolean variables to satisfy the boolean expression that MiniZinc generated from the MiniZinc code shown in the Figure 6.2.

Solvers look for implicit structure in a problem to save them from exhaustive full combinatorial searches. Solvers' performance is typically enhanced by their ability to find and exploit advantageous structure. This is analogous to a rock climbers finding and using cracks as handholds that speed up their assent to the goal of reaching the top.



```
set of int: RANGE = 1..9;
array[RANGE, RANGE] of var RANGE: sudoku;
array[RANGE] of RANGE: first_i = [((k-1) div 3)*3 + 1| k in RANGE]; % first i in
each square
array[RANGE] of RANGE: first_j = [((k-1) mod 3)*3 + 1| k in RANGE]; % first j in
each square


include "alldifferent.mzn";
constraint forall(i in RANGE)(alldifferent([sudoku[i,j] | j in RANGE]));
constraint forall(j in RANGE)(alldifferent([sudoku[i,j] | i in RANGE]));
constraint forall(k in RANGE)(alldifferent([sudoku[i,j] | i in
first_i[k]..first_i[k]+2, j in first_j[k]..first_j[k]+2]));

constraint sudoku[1,1] = 5;
constraint sudoku[1,2] = 3;
constraint sudoku[1,3] = 1;
constraint sudoku[1,6] = 9;
constraint sudoku[1,7] = 6;
constraint sudoku[1,8] = 2;
constraint sudoku[3,6] = 6;
constraint sudoku[3,8] = 9;
constraint sudoku[3,9] = 4;
constraint sudoku[4,2] = 9;
constraint sudoku[4,3] = 6;
constraint sudoku[4,5] = 3;
constraint sudoku[4,6] = 8;
constraint sudoku[4,7] = 1;
constraint sudoku[5,7] = 3;
constraint sudoku[6,1] = 7;
constraint sudoku[6,4] = 6;
constraint sudoku[6,6] = 1;
constraint sudoku[6,8] = 4;
constraint sudoku[7,2] = 6;
constraint sudoku[7,4] = 8;
constraint sudoku[7,7] = 4;
constraint sudoku[8,1] = 1;
constraint sudoku[8,3] = 5;
constraint sudoku[8,5] = 2;

output [show([sudoku[i,j]| j in RANGE]) ++ "\n"| i in RANGE];

solve satisfy;
```

Figure 6.2: Example of MiniZinc code to solve Sudoku puzzle [96].



```
constraint forall(i in RANGE)(alldifferent([sudoku[i,j] | j in RANGE]));
constraint forall(j in RANGE)(alldifferent([sudoku[i,j] | i in RANGE]));
constraint forall(k in RANGE)(alldifferent([sudoku[i,j] | i in
first_i[k]..first_i[k]+2, j in first_j[k]..first_j[k]+2]));
```

Figure 6.3: Three primary Sudoku puzzle constraints expressed succinctly in MiniZinc [96]. The three constraints are no duplication of digits in any row, column, or 3-3 subtable

The example here from the Sudoku puzzle shows the expressive power of MiniZinc to focus succinctly on just the constraints themselves. In contrast to an algorithmic language like C++ which expresses the *how* to solve the problem, the MiniZinc language expresses the *what* to solve. It's up to MiniZinc and its built-in solvers to perform the *how*.

One of the primary goals of this thesis project was to propose the use of satisfiability in a specific geometric way to address automating the arrivals airspace. The following section 6.4 moves on from Sudoku to aviation.

6.4 MiniZinc language - flight paths

Constraint languages, MiniZinc included, can be used to express constraint problems in aviation. In addition to Sudoku, automating aviation is a problem well matched to generic constraint satisfaction machinery. Indeed the constraints formalized as a matrix in Section 5.4 on the Connectivity Matrix, used for expressing descending aircraft flight segment units and their allowable linkages, become the input to MiniZinc satisfiability code.

Figure 6.4 shows a similar matrix to Figure 5.1 in Chapter 5 on Combinatorics, except this is a simpler 2-column from-to matrix. As discussed in that Chapter, the matrix there is a version of an adjacency matrix defining the particular connectivity among flight segments which can be sewn or linked together to form vast numbers of candidate full aircraft-to-runway flight paths.

Figure 6.5 shows the MiniZinc code needed to solve xy-geographic 2 dimen-

```
1, 1063,
1, 1,
1, 7,
1, 6,
1, 8,
2, 10,
3, 78,
4, 81,
4, 80,
...
596, 457,
597, 735,
597, 736,
598, 738,
599, 587,
600, 590,
600, 589,
600, 591,
601, 579,
...
1065, 983,
1066, 986,
1066, 985,
1067, 1056,
985, 0,
1056, 0,
0, 0,
```

Figure 6.4: Matrix enumerating all possible Lego-like flight segments for a 9x9 airspace grid, ultimately encoding the combinatorics of the vast number of ways to construct flight paths. This representation of connectivity is from-to, i.e from one flight segment to another next possible flight segment. For example row 1 column 1 contains flight segment id 1, and row 1 column 2 contains flight segment id 1063. Hence flight segment 1063 is allowed to follow flight segment 1. Matrices expressing connectivity like this range from 1,057 rows for a small 9x9 airspace grid, to 6,996 for a 21x21 airspace grid, to more for larger airspace grids.

sional problem for the optimal flight paths for a suite of aircraft. It's generic code because the same code is used regardless of the particular flight segment connectivity matrix instance read in by the MiniZinc code.

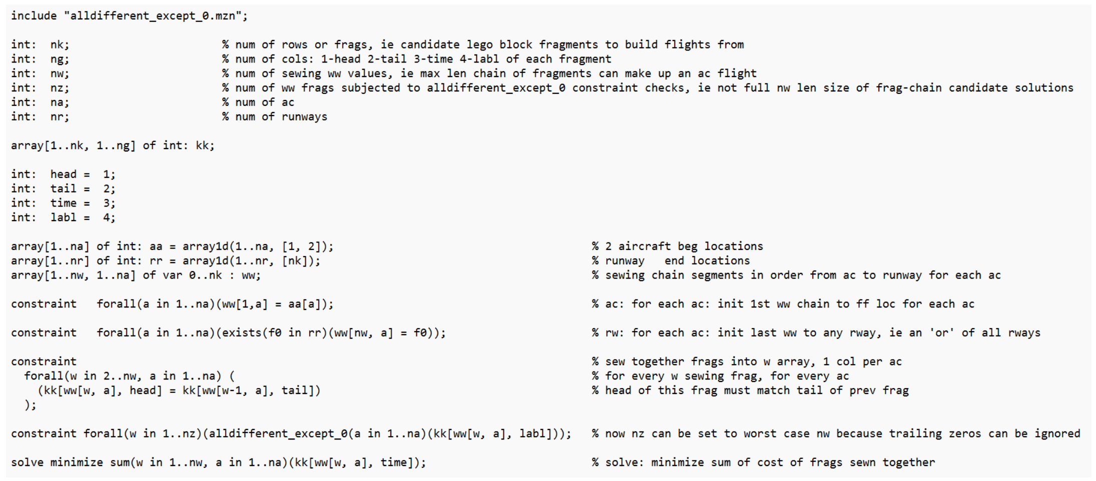
This MiniZinc code (as with the custom solver discussed below in Section 6.7) addresses solving for a safely separated geographical solution, confined to the 2 dimensions of z-altitude and t-time. This is the hard part of solving for the entire 4DT solution, because the additional 2 dimensions of z-altitude and t-time are more deterministic. The hard combinatorial search occurs within 2 geographical z,t dimensions.

However, even with this limited 2-dimensional functional requirement, the computational performance of MiniZinc is still too slow for practical use. Nonetheless, some of the details of the MiniZinc code are discussed below to further explicate this generic constraint satisfaction approach. Details on MiniZinc performance and scalability are also discussed below as well. A custom 2D (and 4DT) solver that runs orders of magnatude faster, indeed fast enough for real-time use, is discussed further below in Section 6.7.

6.4.1 Constraints expressed in MiniZinc

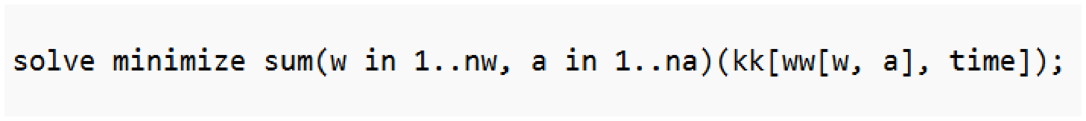
The Sudoku puzzle discussed in Section 6.3 was based on the assumption that there is always a single unique solution to a Sudoku puzzle problem. Therefore, there is no need to optimize the solution since the only solution is also the optimal solution by definition.

For the aviation problem in this thesis project, Figure 6.5 shows the MiniZinc code to express a constraint based solution. There are typically thousands or more sat-



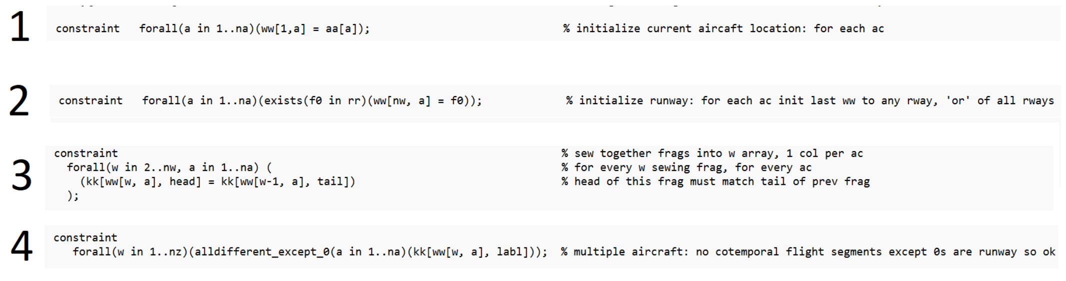
```
include "alldifferent_except_0.mzn";

int:  nk;                       % num of rows or frags, ie candidate lego block fragments to build flights from
int:  ng;                       % num of cols: 1-head 2-tail 3-time 4-labl of each fragment
int:  nw;                       % num of sewing ww values, ie max len chain of fragments can make up an ac flight
int:  nz;                       % num of ww frags subjected to alldifferent_except_0 constraint checks, ie not full nw len size of frag-chain candidate solutions
int:  na;                       % num of ac
int:  nr;                       % num of runways

array[1..nk, 1..ng] of int: kk;

int:  head =  1;
int:  tail =  2;
int:  time =  3;
int:  labl =  4;

array[1..na] of int: aa = array1d(1..na, [1, 2]);                                        % 2 aircraft beg locations
array[1..nr] of int: rr = array1d(1..nr, [nk]);                                          % runway   end locations
array[1..nw, 1..na] of var 0..nk : ww;                                                   % sewing chain segments in order from ac to runway for each ac

constraint   forall(a in 1..na)(ww[1,a] = aa[a]);                                        % ac: for each ac: init 1st ww chain to ff loc for each ac

constraint   forall(a in 1..na)(exists(f0 in rr)(ww[nw, a] = f0));                       % rw: for each ac: init last ww to any rway, ie an 'or' of all rways

constraint                                                                               % sew together frags into w array, 1 col per ac
  forall(w in 2..nw, a in 1..na) (                                                       % for every w sewing frag, for every ac
    (kk[ww[w, a], head] = kk[ww[w-1, a], tail])                                          % head of this frag must match tail of prev frag
  );

constraint forall(w in 1..nz)(alldifferent_except_0(a in 1..na)(kk[ww[w, a], labl]));  % now nz can be set to worst case nw because trailing zeros can be ignored

solve minimize sum(w in 1..nw, a in 1..na)(kk[ww[w, a], time]);                          % solve: minimize sum of cost of frags sewn together
```

Figure 6.5: Sample of MiniZinc code used to calculate an optimal satisfiable suite of descending aircraft paths, xy-geographic 2-dimensional solution only.



```
solve minimize sum(w in 1..nw, a in 1..na)(kk[ww[w, a], time]);
```

Figure 6.6: Solver command instructing MiniZinc to minimize overall cost specified as a sum of costs over all flight segments in solution. See MiniZinc code sample above.



```
1   constraint   forall(a in 1..na)(ww[1,a] = aa[a]);                          % initialize current aircaft location: for each ac

2   constraint   forall(a in 1..na)(exists(f0 in rr)(ww[nw, a] = f0));          % initialize runway: for each ac init last ww to any rway, 'or' of all rways

3   constraint                                                                  % sew together frags into w array, 1 col per ac
      forall(w in 2..nw, a in 1..na) (                                          % for every w sewing frag, for every ac
        (kk[ww[w, a], head] = kk[ww[w-1, a], tail])                             % head of this frag must match tail of prev frag
      );

4   constraint
       forall(w in 1..nz)(alldifferent_except_0(a in 1..na)(kk[ww[w, a], labl]));  % multiple aircraft: no cotemporal flight segments except 0s are runway so ok
```

Figure 6.7: Set of 4 constraints specifying satisfiable solution for flight paths. See MiniZinc code sample above.

isfiable solutions, some closer to optimal than others. Hence, MiniZinc must be further instructed to find the optimal solution across all the satisfiable solutions, expressed according to a formula. In this case, the code searches for a satisfiable solution that *also* minimizes the sum of the costs of the flight segments sewn together into a full flight paths.

Figure 6.6 highlights the last line of the MiniZinc code shown in Figure 6.5. This "solve minimize..." last line instructs MiniZinc in turn to instruct its internal *solver* to search for a satisfiable suite of flight paths that also minimize the sum of flight path costs, as defined in the input matrix file. This is equivalent to shortest path because the cost of each flight segment is defined as 1. For simplicity, Figure 6.4 doesn't show a third column of 1s, but they are there in the full representation of the complete input data to MiniZinc for this satisfiability and optimization problem.

Also, for simplicity of explanation, the "suite" of aircraft is limited to a single aircraft in most of the examples here. Performance metrics discussed below also include solving for 2 aircraft competing for the same runway, in so doing, solving separation conflicts within the MiniZinc run.

Like the Sudoku MiniZinc example, the core of the MiniZinc code is the set 4 constraint statements. Other than the first 2 constraints, effectively used for initialization of aircraft location and runway goal(s), the 2 other constraint statements shown in Figure 6.7 do the core of the work winnowing down the of flyable suite of aircraft flight paths from the current aircraft positions to available runways. These four constraints function as described in the following Subsection 6.4.2.

6.4.2 Four constraints in more detail

This following four bullet items provide a brief explanation of each of the constraints enumerated in Figure 6.7:

1. Initialize current aircraft location: for each aircraft.

2. Initialize runways: for each ac set last element of array ww to a runway value of zero

3. Sew together adjacent flight segments (path segments) tail to head.

4. Disallow same flight segment occupied at the same time, except runways=0

Note that the strategy and style of programming in MiniZinc is quite different from an algorithmic language like C++. Arrays are not used for storing particular data values per se. Rather, arrays in MiniZinc are used for storing a range of *possible* hypothetical values. If the entire implied satisfiability expression is satisfiable, then the arrays have the property that some set of data values exist per all the constraints across all the arrays as specified. Otherwise the result is unsatisfiable or "UNSAT". To draw an analogy with natural language grammar, algorithmic languages express in the indicative mode, while MiniZinc (and similar languages) express in the what-if subjunctive mode.

6.4.3 Unsatisfiability

It is possible that calculating the optimum set of flight paths for some planning cycle, with some state of the airspace, the *solver* engine might return unsatisfiable or UNSAT. This means there exists no set of flight paths that are properly sequenced into

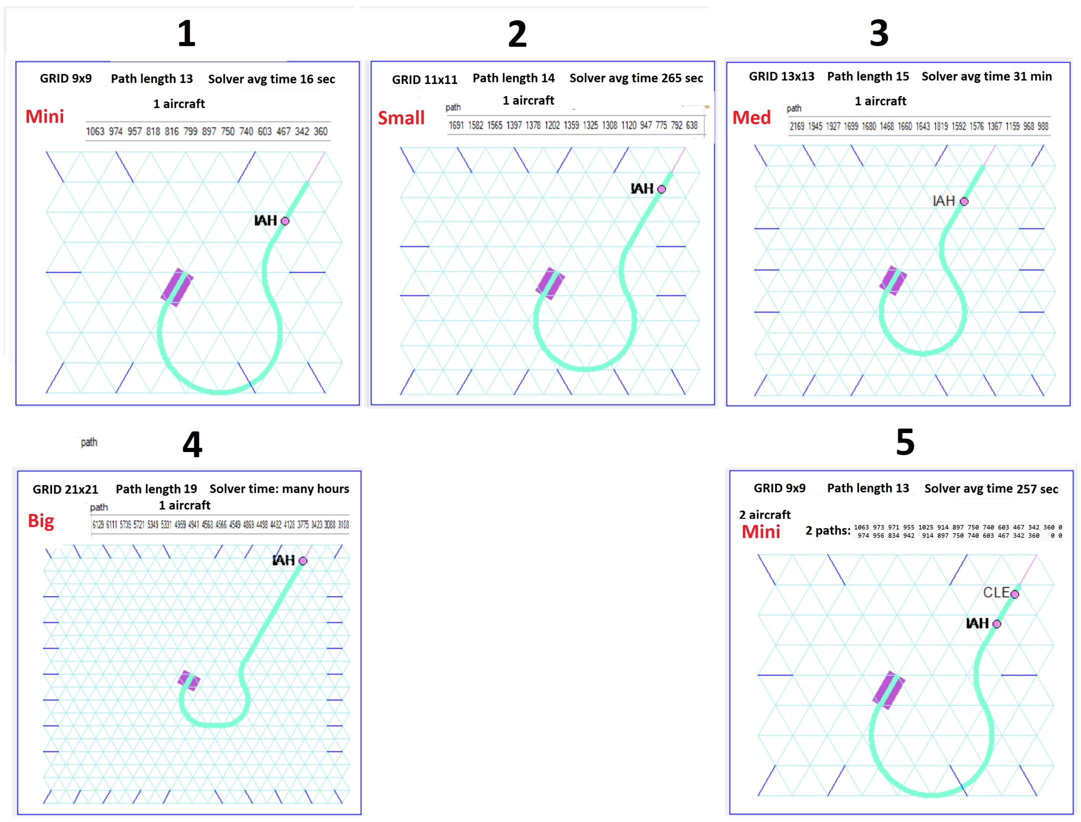


Figure 6.8: Performance tests on different sized airspace grids. Three "mini", "small", "medium" grids are arranged 1,2,3 across the top of the Figure, while a "big" 21x21 grid and a 2-aircraft example are arranged 4,5 across the bottom of the Figure. See performance metrics Table 6.1.

available runways, and contain zero separation conflicts aloft. This means that the local airspace in question is not safely flyable. Obviously, this is a situation that must be strictly avoided. This situation is possible if the airspace is so crowded that safe flight path aircraft optionality is too scarce. This topic is discussed in more detail in upcoming Chapter 7.

Performance	mini	small	med	big	2 ac	inc	inc	inc	inc
Variable	9x9	11x11	13x13	21x21	9x9 2a	9-11	9-13	9-21	ac1-2
Fig6.8 panel	1	2	3	4	5				
grid	9	11	13	21	9	1.22	1.44	2.33	1.00
ac	1	1	1	1	2	1.00	1.00	1.00	2.00
flight segs	1057	2107	3267	6996	1057	1.99	3.09	6.62	1
time secs	16	258	1848	30k+	219	16.1	115	8+hr	13
path length	13	14	15	19	13	1.08	1.15	1.46	1.00
vertices	81	121	169	441		1.49	2.09	5.44	0.00
flatIntVars	4741	8421	13061		9481	1.78	2.75		2.00
flatIntConst	4741	8421	13061		13k	1.78	2.75		2.75
flatTime	1.54	4.53	10.67		3.19	2.93	6.89		2.06
nodes	605	16376	68380		203k	27.1	113		336
failures	303	3635	6322		2897	12.0	20.9		9.56
restarts	1	103	326		159	103	326		159
variables	8m	24m	588m		16m	3.05	7.20		2.00
intVars	4742	8422	13062		9483	1.78	2.75		2.00
boolVars	8m	24m	58m		16m	3.05	7.20		2.00
peakDepth	295	1022	1472		1379	3.46	4.99		4.67
backjumps	6	7416	26661		6858	1236	4443		1143
time	11.5	221.9	1588		206	19.2	137		17.8
initTime	10.9	35.66	64.722		24	3.26	5.92		2.19
solveTime	0.63	186	1523		182	295	2417		288

Table 6.1: MiniZinc performance metrics for a variety of smaller practical grid sizes and number of aircraft. Small step-ups in grid size, etc., result in substantial multiplicative increases (”inc” columns in table here) in solve time. These comparisons are used here to estimate the elapsed time to solve a single aircraft airspace problem with a normal “big” 21x21 grid. The conservative estimate here is that such a larger solution would take MiniZinc more than 8 hours to calculate. Hence, the value of the custom solution discussed below. Note that the tests reported here were performed on a high-end laptop computer Dell XPS 15 (9520) with an i9 Intel processor. Naturally, faster computers would demonstrate faster performance for these problems, but still not likely overcome the poor performance for practical problems as documented here.

6.4.4 Performance tests on MiniZinc solution

For small airspace problems, MiniZinc can be used to calculate correct solutions to these small satisfiability problems. However, for airspaces large enough to be interesting, these MiniZinc solutions do not scale well. Reasonable size problems can take hours to solve. In contrast, the custom solver discussed below in Section 6.7 solves full 21x21 grid airspaces with multiple aircraft in sub-second elapsed time.

To measure MiniZinc performance for various sized airspace grids, 3 sizes of grids were chosen: 9x9, 11x11, and 13x13, here named, "mini", "small", "medium". Even at the "medium" size with one aircraft, MiniZinc computation times were at about half an hour. For the "big" sized 21x21 grids used in airspace simulations, it was impractical to run tests estimated at many hours to calculate.

Figure 6.8 shows 5 test airspace grids. Three "mini", "small", "medium" grids are arranged 1,2,3 across the top of the Figure, while a "big" 21x21 grid and a 2-aircraft example are arranged 4,5 across the bottom of the Figure. These visualizations help inform the performance metrics Table 6.1.

To estimate the scaling properties of MiniZinc for airspace problems discussed in this thesis, a number of MiniZinc runs were performed on various (smaller) sizes of sample airspaces. In addition to MiniZinc providing a mechanism to write out its solution if found, MiniZinc is also fairly generous in outputting internal statistics for each of its runs too. Metrics and some statistics are collected into Table 6.1 showcasing performance comparisons among airspace problem sizes.

As Table 6.1 shows, small increases in airspace grid size (e.g from 9x9 to 11x11) can increase the MiniZinc calculation time by factors of 5 to 16 times. Increasing the number of aircraft from 1 to 2 for the 9x9 grid also increased computation time by over 6 times. Even assuming a conservative doubling of time for each grid size notch-up (e.g from 13x13 to 15x15), the estimated time to solve the "big" 21x21 grid, would be more than 8 hours! And that is for just a single aircraft solution.

Note that all the tests recorded in Table 6.1 were performed on a high-end laptop computer Dell XPS 15 (9520) with an i9 Intel processor. Naturally, faster computers would show faster performance for these problems, but still not likely overcome the poor performance for practical problems as documented here. Improving software technique is indicated.

There is insufficient data collected here to suggest a specific *Big O* formula. There are various parameters which would likely participate in the formula including the size of the path segment connectivity matrix, the length of the optimum solution flight path, and the number of aircraft being deconflicted. As the path length grows by one flight segment, the exponential search combinatorial explosion should increase by a power of 3 corresponding the 3-way left-straight-right optionality at each vertex in the flight path. Also, as size of the path segment connectivity matrix increases, the computation time might also increase proportionately. So, a conservative estimate of a factor of 2 for each notch up from 13x13 to 15x15, etc., would seem reasonable. Some of the internal MiniZinc statistics enumerated in Table 6.1 are also suggestive of poor scaling behavior of this MiniZinc methodology. Hence, the current MiniZinc method for solving the x,y dimensions is not practically scalable at this time.

6.5 MiniZinc internal solvers

The MiniZinc distribution nominally includes 7 different internal *solvers*. Their names are, “Gecode”, “OSICBC”, “Chuffed”, “CPLEX”, “findMUS”, “Globalizer”, “Gurobi”. Three of these (“Gecode”, “OSICBC”, “Chuffed”) worked on small “toy” problems. One solver, “Chuffed”, was by far the fastest solver. Therefore, “Chuffed” was used in all the tests enumerated in Table 6.1.

6.6 Satisfiability problem structure

Even instructing MiniZinc to use a relatively faster solver (“Chuffed”), the MiniZinc code shown in Figure 6.5 is still disappointingly slow for reasonably sized problems. As discussed in Section 6.3, these *solvers* look to find and exploit advantageous internal implicit problem structure to avoid exhaustive NP sized searches. Indeed, constraint problems tend to have underlying structure, which when exploited can dramatically speed up solving the problem. In other words, many constraint problems are not as “NP” as they look.

Given that the “Chuffed” solver was by far the fasted solver tested here, it almost certainly found some useful structure to speed up its combinatorial search process. However, there was still more structure in the problem than the “Chuffed” solver was able to use, as evidenced by the speed of the custom solver discussed in the following Section 6.7 which is 1000s of times faster the MiniZinc code here for this specialized aviation problem.

This raises the question of whether the MiniZinc code shown in Figure 6.5 could

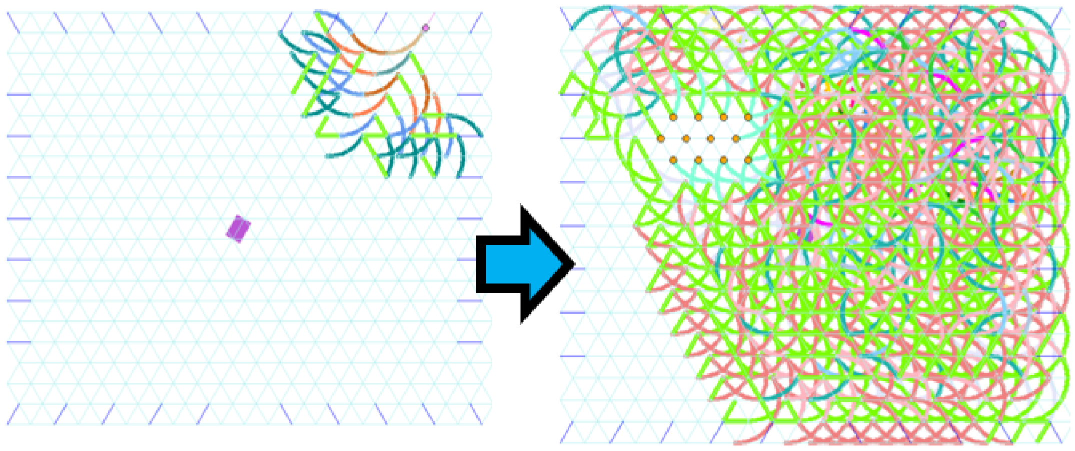


Figure 6.9: Screenshots of exhaustive exploration of all possible flight paths with some important optimizations.

be improved through additional finely targeted constraints to take more advantage of the problem's internal structure. This approach has potential, but is left to future research. For now, use of deeper problem structure is left to the discussion in the following Section 6.7 on custom solver with sub-second performance for reasonably sized problems.

6.7 Custom solver

Although general constraint satisfiability solvers offer convenient, elegant, turn-key solutions, widely available computational performance needed to calculate an optimal solution is not adequate to solve these airspace constraint problems in real time, or even quickly, as noted in Subsection 6.4.4. With available computation getting ever faster and general solver design improving, it's reasonable to expect that solving the local airspace using a general constraint tool will be possible in the future, but not at present. As an alternative for the time being, a custom solver was designed and developed here as part of this thesis project. It offers real-time performance while trading off some quality of optimal solutions. It has been integrated into the computational model simulation discussed here.

6.7.1 Geometry advantages

The underlying geometry of the airspace used here defends against separation conflicts due to the built-in minimum separation of the vertices and their associated flight path segments, as discussed in Section 3.10. The geometry itself greatly simplifies and speeds up the search for safely separated flight paths.

This enables exhaustive search explorations of the triangular grid as illustrated in Figure 6.9 which shows a kind of breadth-first wave exploring connected flight segments outward from a particular aircraft location and heading. This wave will eventually find available runways to land on. In addition to flight segments representing xy-geographical flight segments, these flight segments are also annotated with z-altitude and t-time information, enabling a full 4DT search.

6.7.2 Search by closest aircraft to runway

The basic idea of the custom solver is to begin at the current location of the closest aircraft to a runway, and explore breadth-first all possible paths to an available runway, looking for the shortest (a proxy for the lowest cost) path to a runway. This process is illustrated in Figure 6.9. Once that path has been found, the next closest aircraft is chosen for the succeeding search, and so on.

Clearly this approach will favor the closest aircraft, possibly missing a better solution for the entire suite of aircraft that might be better optimized had a different aircraft had been solved first. In practice, this closest-aircraft method yields very good results because on average the closest aircraft to a runway has the least optionality, so what little optionality there is should be consumed first.

A full satisfiability search would implicitly try all possible searches of all aircraft, with no biases around closest aircraft to runways, but that would stretch computational resources as noted in Subsection 6.4.4.

This custom technique used here for solving the local airspace takes advantage of the specific geometry used to represent the airspace. In particular, vast numbers of redundancies can be skipped. Using a breadth-first exploration reminiscent of the Dijkstra Algorithm for calculating the shortest path through a graph, this custom solver avoids continuing to explore paths that are already longer that the shortest path discovered so far in the search. This vastly reduces the combinatorics of otherwise more exhaustive searches.

6.7.3 Blocking previous paths

Another computational performance optimizing technique uses *blocks*, drawn from operating system design in computer science. Once a best optimal flight path for an aircraft is found, all the vertices of that 4DT path are *blocked* against further consideration in the continuing search process across all aircraft in the local airspace, as illustrated in Figure 6.10. Succeeding path searches will be repelled by existing blocks, and forced to detour around them as illustrated in Figure 6.11.

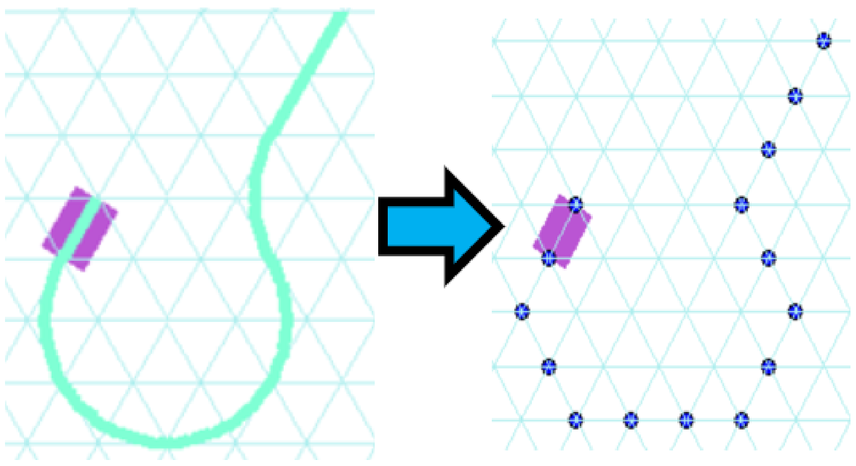


Figure 6.10: Successful flight searches leave behind blocked vertices preventing reuse of these vertices at the same time and nearby altitudes. The left panel shows a smooth flyable representation of the flight path. The right panel shows the vertices corresponding to the smooth flyable flight path.

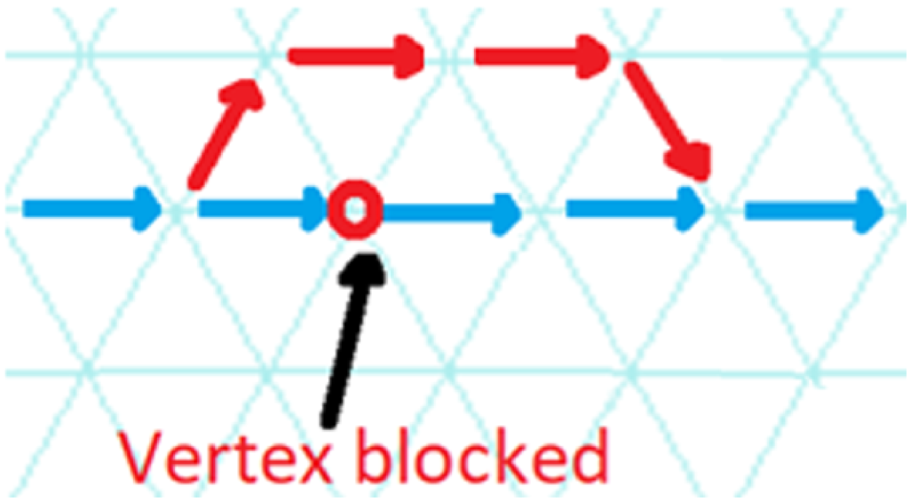


Figure 6.11: Screenshots of exhaustive exploration of all possible flight paths with some important optimizations.

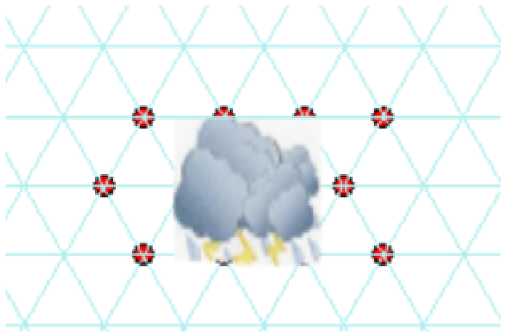


Figure 6.12: Vertices near convective weather are blocked, preventing exploration of flight paths in that area.

6.7.4 Blocking convective weather

Blocked vertices are not only used to avoid simultaneous double use of the same airspace vertex. Vertices are also blocked to prevent exploration of paths through convective weather. Figure 6.12 illustrates how a storm is represented as vertices switched to an “off” state. No flight paths are allowed that would use any of those vertices during a planning cycle. Figure 6.10 shows the same storm vertices avoided by the exploration of flight segments.

6.8 Chapter conclusion

This Chapter 6 discussed methods to find an optimal (or near optimal) solution for an ensemble of arriving aircraft in their descent into available runways.

In principle, constraint satisfaction languages like MiniZinc offer a way to code a general, turn-key method to take a combinatoric enumeration of flight segment connectivity, expressed as a genre of adjacency matrix, to an optimal solution. Unfortunately, computational performance limitations make this convenient, elegant method impractical for the present time, using widely available computational resources.

The alternative method discussed in this chapter is to use the custom solver developed for this thesis research, adapted to the specific geometrical structure of the triangular grid to solve the local airspace in real-time, although sacrificing some quality of the solution. Using this method, the computational model simulator developed for this thesis project demonstrates good results in real-time.

Chapter 7

Conclusion and Future Work

7.1 Summing Up

The purpose of this project is twofold. First, to propose a methodology for automating the arrivals airspace with a free-flight approach utilizing a novel unique geometry coupled with Combinatorics and Satisfiability. The second purpose of this project is to help enable future aviation scientists and engineers to continue this work, so this research direction can be furthered by others.

7.2 Related Ideas and Future Research

7.2.1 Utility Functions

The current optimization expression used in the Satisfiability discussion in Chapter 6 uses a simple utility function expressed in MiniZinc in Figure 7.1. This utility function is a simple sum of the flight costs (flight time) to be minimized over all

```
solve minimize sum(w in 1..nw, a in 1..na)(kk[ww[w, a], time]);
```

Figure 7.1: The current utility function expressed as a MiniZinc command.

the aircraft in the airspace for that solution for that moment in time.

There may be other factors that might be desirable to include in utility functions. For example, the user might wish to minimize turns or 'squirreliness' of the flight paths. Or the user might want to favor certain flights for on-time arrivals to reduce the impact of missed transfer connections. These objectives and others can be included as additive terms in more complex utility functions. In MiniZinc these utility functions would be expressed as a more extensive sum in the "solve minimize" command shown in Figure 7.1.

7.2.2 Fitness Functions

Mathematical evolutionary biology has a convenient formalism for expressing incremental improvements in the viability of biological organisms and species. The *fitness* of an organism (or its species) can be expressed geometrically as a point on a high-dimensional space, where 'higher' points on a fitness "landscape" are more fit, and 'lower' points are less fit. (Note: the vertical dimension of these landscapes is often inverted to correspond to physical units, where greater fitness is expressed as lower energy. But in this discussion here, greater fitness corresponds to a greater value on the vertical axis.)

The goal of an organism is likely to be to improve its "fitness" over time. This can be represented geometrically as ascending up the fitness landscape. If the abstract mathematical landscape is visualized as a simple 3-dimensional mountain range, then the species' goal is to climb up to ever higher mountain peaks.

A simple technique to improve fitness is to travel locally 'up' the mountain landscape. This is called "gradient descent" or ascent. This method fails if the organism gets stuck on a local maxima. Visually, climbing a mountain landscape relying only on local information can get stuck on a low hilltop, missing a nearby Mt. Everest class peak.

Improving an initial set of aircraft routes incrementally over time is similar to landscape fitness problems in biology. Gradient ascent is a simple candidate method to solve this multi-aircraft problem. However, like the organism, this method may get trapped on local maxima. In the case of planning any complex set of aircraft routes, it is almost certain that gradient descent (ascent) algorithms will produce sub-optimal solutions.

7.2.3 Varying Grid Size

The current system as discussed in this thesis uses a universal grid size. The advantage is simplicity. Drawbacks to this uniform geometry are that many candidate shorter or more efficient aircraft routes are missed as the aircraft descend to lower altitudes and to slower speeds where they can make tighter turns than those that are flyable a full speed at higher altitudes.

To address this drawback, one can envision two interlocking grids of different scales. As aircraft descend, they would switch from a course-grained grid to a finergrained grid. Hence the grid could be tessellated at two, or even multiple, tiling sizes. This would be more complicated but might result in increased capacity and efficiency for use of the airspace.

7.2.4 Airspace Capacity, Phase Transitions, Optionality, Unsatisfiability

An important issue in operating an airspace, especially a local arrivals airspace, is how many aircraft should be allowed to enter the local airspace at any given time. More theoretically, when is the airspace “full”? How does the airspace behave near capacity? Are there early warning signs that are precursors to full capacity.

Over-capacity is when the optionality of one or more aircraft is zero. When an aircraft loses optionality, that means that its only options left are violating separation of one or more other aircraft. This is unsafe, so must never occur. From the solver’s point of view, over-capacity means there is no solution that satisfies all the constraints. In this case, the solver returns an unsatisfiability or UNSAT condition.

In Section 6.4.3 on Unsatisfiability, failure to satisfy all the constraints of an airspace is an indication of the loss of optionality in flying the suite of aircraft during the next planning cycle. This means the current flight plans are not properly deconflicted. This is obviously a dangerous condition that must never happen. It’s also an indication of a crowded over-full airspace. An associated research question is can future unsatisfiability be predicted ahead of time. In other words, are there signals that indicate the airspace is becoming too full, before it becomes dangerously full? For any airspace, automated or otherwise, this is an importance area to direct research. Indeed the author and colleagues have done previous work in this area in concert with NASA [87].

Earlier research performed by the author and others [87] demonstrated that there is a phase transition in the airspace from under capacity to over capacity. There are actually two phase transitions: optionality dramatically decreases at the same time as computation time to (attempt to) solve the satisfiability problem dramatically increases.

More research is required to understand and predict these phase transitions well before they might occur. This is critically important in maintaining a high use, yet safe, airspace.

7.2.5 Heterogeneous Aircraft

The current research discussed in this thesis assumes a uniformity of fixed-wing aircraft within some small range of speeds and similar turning profiles. However, many General Aviation (GA) aircraft have slower speed profiles, while rotorcraft have very different turning profiles too. Additional research will be required to operate a truly safe efficient automated airspace with mixed aircraft types.

7.2.6 Genetic And Evolutionary Algorithms

In general, genetic or evolutionary algorithms reportedly show some promise in solving multiple aircraft routing problems. More research is needed here. [77]

7.2.7 Entire Airspace

Although the geometric satisfiability approach described in this thesis is applied only to the arrivals airspace with one airport here, this is fundamentally a general way to automate any airspace, including metroplexes, even the entire airspace. Much research would be required to extend these ideas beyond the narrowly defined airspace addressed in this thesis, but such research looks promising, and worth pursuing in the future.

7.2.8 Lanes

Prof. Adam Smith's has suggested ideas about maintaining stable "lanes" across multiple planning cycles.

The methodology described in this thesis takes a general, yet computation-intensive approach of recalculating the entire airspace for each planning cycle. Being as stateless as possible is simple and elegant in being generally responsive to changing airspace conditions. It also burdens the system with stiff computational requirements for each airspace planning cycle. Since, in practice, the airspace seldom changes abruptly from one minute to the next, another approach could be to maintain a standing set of pre-computed "lanes" available for use for almost any particular configuration of the airspace. Once archived, a particular suite of aircraft at any particular time would only need to initiate a search among some smaller set of a network of "lanes" rather that re-computing the entire airspace from scratch every planning cycle. This approach would need to address occasional abrupt changes in airspace conditions, but perhaps this can be solved by relaxing the need to respond immediately to changing airspace conditions, or other approaches. For automating the entire airspace, this approach appears to have significant merit.

7.2.9 Scaling of Computational Performance

Sections 6.4.4 and 6.6 discuss the poor scaling behavior of the generic Satisfiability solvers tested here. These performance deficiencies suggest future research in the use and design of generalized constraint solvers may someday obviate the need the custom solver described in Section 6.7, in favor of high-performance general solvers. With research progress here, generic solvers may one day form an important basis for the automated airspace of the future.

7.2.10 Limitations

There are many overall issues to be addressed in a fully automated airspace. A few of these issues are location(s) of computation, redundancy of computation, communication needs, failure modes, ghost aircraft, protocols, fully autonomous aircraft fitting in, heterogeneous aircraft including rotor aircraft, exception handling, etc. These are issues that must be addressed before going live with a fully automated airspace.

7.2.11 UCSC Autonomous Systems Lab

Many labs including Prof. Gabriel Elkaim's Autonomous Systems Lab (ASL) are researching aviation and related areas. This current thesis fits within a context of much ongoing research in aviation and other related areas. Much to do.

7.3 Future Publications

The research described in this thesis may be of interest to other researchers in the field, in the form of future academic papers. Candidate publications and conferences for describing this research in the future are as follows. Note that the author of this thesis has previously published in the first two of the three publications listed below. Much thought will need to go into choosing the best way to further the research in this thesis.

1. ATIO - Proceedings of the Annual Aviation Technology, Integration, and Operations Conference. [86] [43] [44]

2. ATM conference proceedings - USA/Europe Air Traffic Management Research and Development Seminar. [87]

3. AAMAS conference proceedings - International Conference on Autonomous Agents and Multiagent Systems. [1]

7.4 End game

Figure 7.2 illustrates the end game of automating the airspace: the system only requires one human and one dog to run the system. The dog is there to bite the human in case the human tries to do anything [88].

Figure 7.2: The End Game – Future automated airspace. The system only requires one human and one dog to run the system. The dog is there to bite the human in case the human tries to do anything [88].

7.5 Concluding thoughts

The author looks forward to continuing this research in refining the architecture of safe efficient automated airspaces of the future based on free-flight, built upon some of the ideas and principles discussed here in this thesis. Appreciations to all who have contributed and guided this work.

Bibliography

[1] AAMAS-22, 2022, Tutorial on Recent Advances in Multi-agent path finding (MAPF) [ref by Prof Adam Smith], https://www.youtube.com/watch?v=H3wRCZf_Mrs

[2] AAMAS-22 Airport Surface Operations, 2022, Tutorial on Recent Advances in Multi-agent path finding (MAPF) [ref by Prof Adam Smith], https://youtu.be/H3wRCZf_Mrs?t=550

[3] AAMAS-22 Evaluating plan quality, 2022, Tutorial on Recent Advances in Multi-agent path finding (MAPF) [ref by Prof Adam Smith], https://youtu.be/H3wRCZf_Mrs?t=6700

[4] AAMAS-22 Extensions, 2022, Tutorial on Recent Advances in Multi-agent path finding (MAPF) [ref by Prof Adam Smith], https://youtu.be/H3wRCZf_Mrs?t=3998

[5] AAMAS-22 How prioritized planning fails, 2022, Tutorial on Recent Advances in Multi-agent path finding (MAPF) [ref by Prof Adam Smith], https://youtu.be/H3wRCZf_Mrs?t=6786

[6] AAMAS-22 Large neighborhood search, 2022, Tutorial on Recent Ad-

vances in Multi-agent path finding (MAPF) [ref by Prof Adam Smith], https://youtu.be/H3wRCZf_Mrs?t=7053

[7] AAMAS-22 List of domains where MAPF problems arise, 2022, Tutorial on Recent Advances in Multi-agent path finding (MAPF) [ref by Prof Adam Smith], https://youtu.be/H3wRCZf_Mrs?t=7518

[8] AAMAS-22 Pipe Routing, 2022, Tutorial on Recent Advances in Multi-agent path finding (MAPF) [ref by Prof Adam Smith], https://youtu.be/H3wRCZf_Mrs?t=635

[9] AAMAS-22 22-Reduction solvers, 2022, Tutorial on Recent Advances in Multi-agent path finding (MAPF) [ref by Prof Adam Smith], https://youtu.be/H3wRCZf_Mrs?t=1558

[10] AAMAS-22 Rolling Horizon Collision Resolution, 2022, Tutorial on Recent Advances in Multi-agent path finding (MAPF) [ref by Prof Adam Smith], https://youtu.be/H3wRCZf_Mrs?t=6943

[11] AAMAS-22 Search-based suboptimal solvers, 2022, Tutorial on Recent Advances in Multi-agent path finding (MAPF) [ref by Prof Adam Smith], https://youtu.be/H3wRCZf_Mrs?t=1220

[12] Achlioptas, D. and Coja-Oghlan, A., 2008, October. Algorithmic barriers from phase transitions. In Foundations of Computer Science, 2008. FOCS'08. IEEE 49th Annual IEEE Symposium on (pp. 793-802). IEEE.

[13] Agarwal, P.K., Raghavan, P. and Tamaki, H., 1995, May. Motion planning for a

steering-constrained robot through moderate obstacles. In Proceedings of the twenty-seventh annual ACM symposium on Theory of computing (pp. 343-352). ACM.

[14] Airbus flight manual 2018, Descent graphic. Descent Performance Versus Cost Index, page 51

[15] Aircraft Owners and Pilots Association, 2017. Sensing Winds Aloft, https://www.aopa.org/news-and-media/all-news/2017/september/flight-training-magazine/how-it-works-winds-aloft.

[16] Arad, B.A., The control load and sector design, 1964, Journal of Air Traffic Control

[17] Aircraft Situation Display to Industry (ASDI) data, August 29, 2013, data base of "a day in the life of the NAS" available from the FAA and from the author.

[18] Babel, L., 2013. Three-dimensional route planning for unmanned aerial vehicles in a risk environment. Journal of Intelligent & Robotic Systems, 71(2), pp.255-269.

[19] Base of Aircraft Data (BADA). Eurocontrol (European Organization for the Safety of Air Navigation) proprietary database. http://www.eurocontrol.int/services/bada

[20] Bailey, D.D., Dalmau, V. and Kolaitis, P.G., 2007. Phase transitions of PP-complete satisfiability problems. Discrete Applied Mathematics, 155(12), pp.1627-1639.

[21] Beginner's Guide to Aviation Efficiency, ATAG 2010

[22] Beigel, R., Hemachandra, L.A. and Wechsung, G., 1989. On the power of probabilistic polynomial time. Johns Hopkins University. Department of Computer Science.

[23] Bicchi, A. and Pallottino, L., 2000. On optimal cooperative conflict resolution for air traffic management systems. Intelligent Transportation Systems, IEEE Transactions on, 1(4), pp.221-231.

[24] Boeing, Fuel Conservation Strategies, 2010, Boeing Aero Magazine, 2nd quarter 2010

[25] Boissonnat, J.D. and Lazard, S., 1996, May. A polynomial-time algorithm for computing a shortest path of bounded curvature amidst moderate obstacles. In Proceedings of the twelfth annual symposium on Computational geometry (pp. 242-251). ACM.

[26] bts.gov 2016 - Airline Fuel Cost and Consumption (U.S. Carriers - Scheduled) January 2000 - February 2016 http://www.transtats.bts.gov/fuel.asp

[27] Chen Jing-Chao, 2003 - Dijkstra's Shortest Path Algorithm

[28] Chitsaz, H. and LaValle, S.M., 2007, December. Time-optimal paths for a Dubins airplane. In Decision and Control, 2007 46th IEEE Conference on (pp. 2379-2384). IEEE.

[29] Croydon Airport, Transport Heritage, 2015, "Heritage Locations – South East – Surrey – Croydon Airport"..

[30] Curry, Ren E., private conversations and correspondence. (cf. [Bell 1995, Curry 1996]

[31] Curry, R.E., and Wagner, G.A., "Highly-Optimized Flight Plans in Today's Airline Environment", AGIFORS 36th Symposium, Atlanta, 1996

[32] Dijkstra, E.W., 1959. A note on two problems in connexion with graphs. Numerische mathematik, 1(1), pp.269-271.

[33] Dubins, L. E. (1957). "On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents". American Journal of Mathematics. 79 (3): 497. doi:10.2307/2372560. JSTOR 2372560.

[34] Erzberger, H., Paielli, R.A., Isaacson, D.R. and Eshow, M.M., 1997, June. Conflict detection and resolution in the presence of prediction error. In 1st USA/Europe Air Traffic Management R&D Seminar, Saclay, France (pp. 17-20).

[35] Erzberger H., private conversations.

[36] FAA Historical Chronology, 1926-1996, Federal Aviation Administration

[37] FAA 2016 - Automatic Dependent Surveillance-Broadcast (ADS-B), https://www.faa.gov/nextgen/programs/adsb/

[38] FAA Briefing for SFO, 2018, FAA Briefing for SFO 2018.pdf

[39] FAA Aeronautical Charts, flight maps of Standard Terminal Arrival Routes (STARs) available from www.faa.gov

[40] FAA Air Traffic By The Numbers, www.faa.gov/air_traffic/by_the_numbers

[41] FAA ADS-B Automatic Dependent Surveillance-Broadcast, https://www.faa.gov/air_traffic/technology/adsb

[42] Garcia, Prett, Morari, 1989, Model Predictive Control Theory and Practice a Survey. International Federation of Automatic Control

[43] Gawdiak, Holmes, B., Sawhill, B., Herriot, J., et al., 2012, Air Transportation Strategic Trade Space Modeling and Assessment Through Analysis of On-Demand Air Mobility with Electric Aircraft, Proceedings of the 12th Annual ATIO Conference, Indianapolis, IN.

[44] Gawdiak, Y., Herriot, J., Sawhill, B., et al., 2012, Modeling of Demand and Supply for Air Transportation in the U.S., 2025 - 2040, Proceedings of the 12th Annual ATIO Conference, Indianapolis, IN.

[45] Goerzen, C., Kong, Z. and Mettler, B., 2010. A survey of motion planning algorithms from the perspective of autonomous UAV guidance. Journal of Intelligent & Robotic Systems, 57(1-4), pp.65-100.

[46] Google images, 2018. Search term "Engineering Control Theory" and "open-loop control".

[47] GrandCanyonCollision, artist's conception. Google images and www.lostflights.com.

[48] Hamming, R. W., 1950, Error Detecting and Error Correcting Codes. The Bell System Technical Journal Vol. XXIX, April 1950.

[49] Hart, P.E., Nilsson, N.J. and Raphael, B., 1967. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. Stanford Research Institute.

[50] Hemachandra, L.A. and Wechsung, G., 1988. On the Power of Probabilistic Polynomial Time: PNP [log] PP.

[51] Hoare, Sir Tony. The Fallacy of Premature Optimization (article) ACM, Ubiquity, Volume 2009 Issue February, https://ubiquity.acm.org/article.cfm?id=1513451

[52] Hoekstra, J.M., van Gent, R.N. and Ruigrok, R.C., 2002. Designing for safety: the 'free flight' air traffic management concept. Reliability Engineering & System Safety, 75(2), pp.215-232.

[53] Holmes, B., Sawhill, B., Herriot, J. and Seehart, K., 2012, Development of Complexity Science and Technology Tools for NextGen Airspace Research and Applications, NASA Technical Report NASA/CR-2012-217580

[54] Hota, S. and Ghose, D., 2009, June. A modified Dubins method for optimal path planning of a miniature air vehicle converging to a straight line path. In American Control Conference, 2009. ACC'09. (pp. 2397-2402). IEEE.

[55] Hwang, Y.K. and Ahuja, N., 1992. Gross motion planning¬ a survey. ACM Computing Surveys (CSUR), 24(3), pp.219-291.

[56] Jaillet, L., Cortés, J. and Siméon, T., 2008, September. Transition-based RRT for path planning in continuous cost spaces. In Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on (pp. 2145-2150). IEEE.

[57] Jardin, M.R., 2003. PhD dissertation, Stanford University

[58] Jardin, M.R., 2003. Toward real-time en route air traffic control optimization.

[59] Khadilkar, H. and Balakrishnan, H., 2016. Integrated Control of Airport and Terminal Airspace Operations. Control Systems Technology, IEEE Transactions on, 24(1), pp.216-225.

[60] Kim, J. and Hespanha, J.P., 2003, December. Discrete approximations to continuous shortest-path: Application to minimum-risk path planning for groups of UAVs. In Decision and Control, 2003. Proceedings. 42nd IEEE Conference on (Vol. 2, pp. 1734-1740). IEEE.

[61] Kirkpatrick, S. and Selman, B., 1994. Critical behavior in the satisfiability of random Boolean expressions. Science, 264(5163), pp.1297-1301.

[62] Knuth D., 1984, The TeXbook, Addison-Wesley Professional, ISBN-10: 0201134489, ISBN-13: 978-0201134483

[63] Knuth D., 2015, The Art of Computer Programming, Volume 4, Fascicle 6, Satisfiability

[64] Leslie Lamport, *LaTeX: a document preparation system*, Addison Wesley, Massachusetts, 2nd edition, 1994.

[65] LaValle, S., 2006. Planning Algorithms, Cambridge Univ.

[66] MAPF Multi-agent pathfinding problem, 2023, Wikipedia article on Multi-agent path finding (MAPF) [ref by Prof Adam Smith], https://en.wikipedia.org/wiki/Multi-agent_pathfinding

[67] LaValle Steven M, MAPF online textbook on planning algorithms, 2020, textbook) [ref by Prof Adam Smith], http://lavalle.pl/planning/web.html

[68] LaValle Steven M, Prioritized planning, 2020, article [ref by Prof Adam Smith], http://lavalle.pl/planning/node324.html

[69] Mathews, G. B., On the Partition of Numbers, 1896, Proceedings of the London Mathematical Society

[70] Maxwell James Clerk, 2018, Governors and Feedback Control, clerkmaxwellfoundation.org

[71] Mead 2007 - Tailored Arrivals, Boeing, Rob Mead, 9 Jan 2007, on file

[72] Menon, P.K., Sweriduk, G.D. and Sridhar, B., 1999. Optimal strategies for free-flight air traffic conflict resolution. Journal of Guidance, Control, and Dynamics, 22(2), pp.202-211.

[73] Minizinc the language: https://www.minizinc.org/

[74] Miquel, Thierry, Path Stretching and Tracking for Time-Based Aircraft Spacing at Meter Fix, 2006, The American Institute of Aeronautics and Astronautics (AIAA).

[75] Mogford, R. H. ; Guttman, J. A. ; Morrow, S. L. ; Kopardekar, P, The Complexity Construct in Air Traffic Control: A Review and Synthesis of the Literature, 1995, Defense Technical Information Center.

[76] NACTA A History of Air Traffic Control, 2002, National Air Traffic Controllers Association

[77] Nikolos, I.K., Valavanis, K.P., Tsourveloudis, N.C. and Kostaras, A.N., 2003. Evolutionary algorithm based offline/online path planner for UAV navigation. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 33(6), pp.898-912.

[78] OAG data, 1989-2010, data base available from the OAG and from the author

[79] Paielli, R.A. and Erzberger, H., 1997. Conflict probability for free flight. Journal of Guidance, Control, and Dynamics, 20(3), pp.588-596.

[80] Panait, L. and Luke, S., 2005. Cooperative multi-agent learning: The state of the art. Autonomous Agents and Multi-Agent Systems, 11(3), pp.387-434.

[81] Pechoucek, M. and Sislak, D., 2009. Agent-based approach to free-flight planning, control, and simulation. Intelligent Systems, IEEE, 24(1), pp.14-17.

[82] Range Performance Aero, 2018. Cost Index graphic. Aviation website, code7700.com

[83] Rios, L.H.O. and Chaimowicz, L., 2010. A survey and classification of A* based best-first heuristic search algorithms. In Advances in Artificial Intelligence-SBIA 2010 (pp. 253-262). Springer Berlin Heidelberg.

[84] Rios, L.H.O. and Chaimowicz, L., 2011. PNBA*: A Parallel Bidirectional Heuristic Search Algorithm. In Anais do XXXI Congresso da Sociedade Brasileira de Computação – VIII Encontro Nacional de Inteligência Artificial (ENIA) (Vol. 287).

[85] Salkin, Harvey - The Knapsack Problem, A Survey, 1975, Case Western Reserve University

[86] Sawhill, B., Herriot, J., and Holmes, B.J., 2011, Complexity Science Tools for Interacting 4D Trajectories and Airspace Phase Transitions, Proceedings of the 11th Annual ATIO Conference, Virginia Beach, VA

[87] Sawhill, B., Herriot, J., Holmes, B.J., and Seehart, K., 2011, Airspace Phase Transitions and the Traffic Physics of Interacting 4D Trajectories, Proceedings of the Ninth USA/Europe Air Traffic Management Research and Development Seminar (ATM 2011), Berlin, Germany (best paper award)

[88] Sawhill B., 2016-2018 - private conversations and correspondence

[89] Schoemig, Ewald G.; Armbruster, John; Boyle, Daniel; Haraldsdottir, Aslaug; Scharl, Julien, 3D Path Concept and Flight Management System (FMS) Trades, 2006, ieee aiaa 25TH Digital Avionics Systems Conference

[90] Seenivasan Dineh et al, 2019, Multiaircraft Optimal 4D Trajectory Planning Using Logical Constraints

[91] Shanmugavel, M., Tsourdos, A., White, B. and bikowski, R., 2010. Co-operative path planning of multiple UAVs using Dubins paths with clothoid arcs. Control Engineering Practice, 18(9), pp.1084-1092.

[92] SPPA 2016 - Sky Posse Palo Alto http://www.skypossepaloalto.org/

[93] Stackexchange: How to calculate angular velocity and radius of a turn https://aviation.stackexchange.com/questions/2871/how-to-calculate-angular-velocity-and-radius-of-a-turn

[94] Stone, P. and Veloso, M., 2000. Multiagent systems: A survey from a machine learning perspective. Autonomous Robots, 8(3), pp.345-383.

[95] Suchkov, A., Swierstra, S. and Nuic, A., 2003, June. Aircraft performance modeling

for air traffic management applications. In 5th USA/Europe Air Traffic Management Research and Development Seminar (pp. 23-27).

[96] Minizinc example code for expressing and solving Soduko puzzle, from github: https://github.com/buzzdecafe/minizinc/blob/master/sudoku.mzn

[97] Sujit, P.B., Saripalli, S. and Borges Sousa, J., 2014. Unmanned aerial vehicle path following: A survey and analysis of algorithms for fixed-wing unmanned aerial vehicless. Control Systems, IEEE, 34(1), pp.42-59.

[98] Tomlin, C., Pappas, G.J. and Sastry, S., 1998. Conflict resolution for air traffic management: A study in multiagent hybrid systems. Automatic Control, IEEE Transactions on, 43(4), pp.509-521.

[99] Luca Vigano, Marco Bergamasco, Marco Lovera and Andras Varga, 2009, Optimal periodic output feedback control: a continuous-time approach and a case study

[100] Wang, C., Soh, Y.C., Wang, H. and Wang, H., 2002. A hierarchical genetic algorithm for path planning in a static environment with obstacles. In Electrical and Computer Engineering, 2002. IEEE CCECE 2002. Canadian Conference on (Vol. 3, pp. 1652-1657). IEEE.

[101] Waslander, S.L., Raffard, R.L. and Tomlin, C.J., 2006, June. Toward efficient and equitable distributed air traffic flow control. In American Control Conference, 2006 (pp. 6-pp). IEEE.

[102] P. Wei a, L. Chen b, D., 2014, Sun Algebraic connectivity maximization of an air transportation network The flight routes' additiondeletion problem

[103] Wickens, C.D., Hellenberg, J. and Xu, X., 2002. Pilot maneuver choice and workload in free flight. Human Factors: The Journal of the Human Factors and Ergonomics Society, 44(2), pp.171-188.

[104] Wolfram, 2018. Calculus and Analysis; Dynamical Systems. http://mathworld.wolfram.com/DynamicalSystem.html

[105] Zuniga, C.A.; Piera, M.A.; Ruiz, S.; Del Pozo, I., A CD&CR causal model based on path shortening/path stretching techniques, 2011, Transportation Research Part C 33 (2013) 238–256