Optimal Air Traffic Control automation: application to oceanic control

Francisco Freitas
Mestrado em Engenharia Aeroespacial
Instituo Superior Técnico

Abstract — ATCO workload limits impose upper bounds to the amount of traffic manageable in a given air sector for a given time frame. Therefore, ATC automation methods open the possibility of reducing this workload by shifting to the machine the tasks of (1) detecting potential conflicts in long-term, and of (2) proposing ATC instructions to the operator that prevent such conflicts. We propose a decision support system based on a combinatorial optimization approach using a branch-and-bound method. Given a known traffic situation, we proceed by simulating the long-term trajectories of traffic, taking into account possible instructions to separate traffic. Uncertainty is modeled assuming wind as the only non-deterministic source. As a case study, we analyse the problem of oceanic airspace, where conventional ATC is (mostly) used due to the lack of radar coverage. In this study we considered only flight level change instructions, given at report fixes only. The cost function employed includes both a measure of vertical deviation from the filed flight plan (FPL) and the total amount of ATC instructions. The multi-criteria problem is solved by inserting the air traffic controller in the optimization loop through an interactive decision making configuration. The algorithm is to be run every time a position report from an aircraft is received.

Keywords: ATC automation; Conflict detection and resolution; branch-and-bound; interactive decision-making; oceanic control

1 Introduction

The current Air Traffic Management system is already near its full capacity in some of the most busy sectors, and will eventually be overloaded if air travel grows as predicted [9]. Given that human agents (ATCO) are already working at the top of their capacities and that an increase in their workload would most likely be a threat to system safety, an increase in air traffic can only be safely handled by a future ATC system if solutions are found to reduce their workload. Several paradigms have been studied to tackle the problem.

A commonly studied approach focuses on redesigning the existing airspace organization, by creating functional airspace blocks designed to simplify air traffic control and by allowing aircrafts to fly direct routes between fixes (Free Route). This concept is being developed in Europe, under SESAR, but still in a research phase. Frequently connected with this approach is the area of study concerned with the decentralization of ATC, which is based on passing some of the ATCO’s tasks to aircrafts, allowing agents to self-organize themselves and to perform conflict detection and resolution autonomously. Such system would have to rely on sophisticated and sound airborne conflict detection and resolution systems, and seems difficult to implement in the next few years. A study comparing the performances of centralized and decentralized strategies is presented in [11]. Other contributions in this area may be found in [2], [4], [8] and [10].

A less dramatic depart from today systems, and better suited for short-term application, is the approach of developing computerized decision support tools to improve the performance of the cur-
rent centralized system. Solutions derived from this approach may range from simple tools, that detect short and medium-term conflicts without suggesting any solutions (already being broadly used), to total automation, in which an automatic tool detects conflicts, searches for an optimal solution and issues the instructions to the aircrafts by means of some form of digital data format, replacing human controllers.

Several solutions were presented in this area. The studies described in [3] and [14] use Genetic Algorithms to find resolution maneuvers. Chiang et. al describe a Computational Geometry methodology to calculate conflict-free routes in [1]. In [6] and [5], 3D-trajectories are allocated using A* and Genetic Algorithms. Optimal Control techniques are used in [13] to perform conflict detection and resolution.

Our work follows this latter paradigm, and lies midway between a passive system and total automation.

1.1 Approach outline

The developed program receives as inputs the current situation of a set of aircrafts and their filed FPLs inside a certain flight region, and calculates their long-term predicted trajectories within a certain time window, using a point-mass dynamics model and a simplified Autopilot. A combinatorial search is carried out to analyse each plan within the state-space. Pairwise conflict detection is executed for each individual plan. A Cost Function taking into account the deviation from the filed FPL and the total amount of ATC instructions was created, allowing a branch-and-bound method to be implemented, pruning regions of the solution search and reducing simulation time, assuring optimality. A Monte Carlo simulation is ran as a robustness check to test each calculated plan. A set of particles is created for each aircraft, simulating possible trajectories. At the end of the simulation, the algorithm returns a global plan, consisting of a set of instructions to be issued to each aircraft, and presents this plan to the human controller. The ATCO may accept this suggestion or request the algorithm to recalculate, introducing a new restriction to the optimization.

As a case study, we chose to analyse the operation in Oceanic Airspace, where conventional ATC is used due to the lack of radar coverage. Restrictions were created so that aircrafts fly their filed horizontal route at the requested airspeed, and trajectory changes are only allowed in the vertical plane, with instructions being issued only at report fixes. The algorithm is run every time a position report is received from an aircraft, ensuring the current advisory is based on the most updated information available.

The structure of this paper is as follows. The dynamical model used to predict future aircraft trajectories is described in section 2, where the main equations defining an aircraft motion are presented and details are given about the position update routine and the way in which uncertainty is modelled. Section 3 describes the branch-and-bound search algorithm that performs conflict detection and resolution. Relevant experimental results are presented at section 4. The main conclusions of this paper and a description of future work may be found in section 5.

2 Trajectory Prediction

The method for the prediction of the future trajectory of an aircraft is based on the one presented by Glover and Lygeros in [7]. Having at its disposal the current state of an aircraft – position, speed and attitude – its future intention – in FPL format and an aircraft-specific model – loaded from Eurocontrol’s database BADA – the program is able to predict the trajectory of that aircraft using a point-mass model. A six-state control system is implemented, with its states being: horizontal position ($x_1$ and $x_2$), altitude ($x_3$), true airspeed ($x_4$), heading angle ($x_5$) and mass ($x_6$). The aircrafts are assumed to have three control inputs: engine thrust ($u_1$), bank angle ($u_2$) and flight path angle ($u_3$). The differential equations defining this model are:

\[
\begin{align*}
\dot{x} &= \begin{pmatrix}
  x_4 \cos(x_5) \cos(u_3) + w_1 \\
  x_4 \sin(x_5) \cos(u_3) + w_2 \\
  x_4 \sin(u_3) + w_3 \\
  \frac{C_D S \rho x_2}{2} - g \sin(u_1) + \frac{1}{x_5} u_1 \\
  \frac{L S \rho x_2}{2} x_6 \sin(u_2) \\
  -\eta u_1
\end{pmatrix}
\end{align*}
\]

where $x = [x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6]^{\top}$, $\rho$ is the local air density (varying with altitude, thus dependent on
$x_3$). $C_L$ is the lift coefficient of the aircraft, $C_D$ its drag coefficient, $S$ its wing area, and $\eta$ the thrust specific fuel consumption. These four parameters are calculated using coefficients read from BADA files for each specific aircraft model and applying the equations presented on BADA’s User Manual [15].

The equations representing the speed in each of the three coordinate axes ($\dot{x}_1, \dot{x}_2$ and $\dot{x}_3$) all include a factor representing a component of the wind vector $W = (w_1, w_2, w_3)$. Details on uncertainty modeling are given in subsection 2.3.

The three control inputs form a basic Autopilot, ensuring the aircraft follows the trajectory and airspeed required by its FPL. These control inputs result from proportional-integral controllers for the Thrust and the Flight Path Angle in Climb/Descent phase, and from proportional controllers for the Bank Angle and the Flight Path Angle in Cruise phase. The equations defining the Autopilot are:

$$u_1(t) = k_{p1}e_v(t) - k_{i1}\int_0^t e_v(\tau)d\tau$$
$$u_2(t) = k_{p2}\delta(t) + k_{p3}\theta(t)$$
$$u_3(t) = \begin{cases} k_{p4}e_{rocd}(t) + k_{i2}\int_0^t e_{rocd}(\tau)d\tau & \text{C/D} \\ k_{p5}e_h(t) & \text{CRZ} \end{cases}$$

where $e_v(t) = v_{ref} - v(t)$ is the true airspeed error, $e_{rocd}(t) = ROCD_{ref} - ROCD(t)$ is the vertical speed error while performing maneuvers, $\delta(t)$ is the cross track deviation from the reference path and $\theta(t) = \varphi_{ref} - \varphi(t)$ is the heading error. C/D and CRZ denotes the climb/descend and cruise flight phases.

This continuous-time system is discretized using a fourth-order Runge-Kutta method to solve the ordinary differential equations. From now on, system variables are indexed with a discrete index $k$, i.e. $t = kT$, where $T$ is the integration time step.

2.1 Spherical Coordinates using Quaternion Algebra

An important change had to be made to the above model: being expressed in a cartesian orthogonal frame, it is unsuited to deal with large length trajectories, as it does not account for the curvature of the Earth. This limitation was overcome using quaternion algebra to calculate each successive position of an aircraft by applying a 3D rotation about an axis containing the center of the Earth to the aircraft position at the last time step. This way, the aircraft horizontal position ceases to be expressed as a $(x, y)$ pair, and the state variables $x_1$ and $x_2$ become geographic Latitude ($\varphi$) and Longitude ($\lambda$).

In this model, each position over the Earth surface can be represented by a rotation quaternion, in the following manner. We consider a $\mathbb{R}^3$ vector $p$, with $||p|| = R_E$ (where $R_E$ denotes the Earth radius), with its origin at the center of the Earth, and pointing to the North Pole, i.e. $p$ may be expressed in an Earth-Centered Earth-Fixed (ECEF) coordinate system as $p_{ECEF} = (0, 0, R_E)$. A position $A$, over the Earth surface, with geographic coordinates $(\varphi, \lambda)$ may be uniquely represented by a rotation of vector $p$ into another vector $p'$ that points to position $A$, as shown in figure 1(a) (where the represented frame is ECEF).

This rotation is fully defined by an axis of rotation $k$ – which lies on the equatorial plane – and by

![Figure 1: Rotation parameters representation](image-url)
A rotation angle $\theta$. Two planes are considered to find the rotation parameters: the equatorial plane (shown in figure 1(b)), and a vertical plane (perpendicular to the equatorial plane) which contains vector $p$ and point $A$ (shown in figure 1(c)).

A quick inspection of figure 1(b) reveals vector $k$ to point to a longitude of $\lambda + \frac{\pi}{2}$ and thus its coordinates are $(R_T \cos (\lambda + \frac{\pi}{2}), R_T \sin (\lambda + \frac{\pi}{2}), 0)$, whereas 1(c) shows that the rotation angle depends solely on point $A$’s latitude: $\theta = \frac{\pi}{2} - \varphi$.

Following the method explained in [12], this rotation may then be represented as a quaternion:

$$q_P = \frac{u_x \sin \frac{\theta}{2}}{u_y \sin \frac{\theta}{2}}$$

where the index $P$ indicates a position quaternion, $u = (u_x, u_y, 0)$ = $\frac{k}{||k||}$ is a unit axis of rotation and $\theta$ is the rotation angle. This quaternion rotates $p$ into $p'$ using the following relation:

$$p' = (q_P^*)p(q_P)$$

where $q_P^*$ is the complex conjugate of quaternion $q_P$.

### 2.2 Position Update

At some iteration $k$ of the algorithm, the aircraft is at geographical position $(\varphi[k], \lambda[k])$, equivalent to some position quaternion $q_P[k]$, flying at altitude $h$, at True Airspeed $v$.

Each aircraft’s Autopilot constantly adjusts its heading angle $\psi$ so that it follows a Great Circle path between its current position $(\varphi_i, \lambda_i)$ and its destination $(\varphi_f, \lambda_f)$, according to the formula:

$$\psi = \arctan2(\sin(\lambda_f - \lambda_i) \cos(\varphi_f), \cos(\varphi_i) \sin(\varphi_f) - \sin(\varphi_i) \cos(\varphi_f) \cos(\lambda_f - \lambda_i))$$

Assuming leveled flight, and thus zero vertical speed, it is possible to obtain $v_{\text{NED}}$, the aircraft’s speed vector expressed in a North-East-Down local frame:

$$v_{\text{NED}} = (v \cos \psi, v \sin \psi, 0)$$

$v_{\text{ECEF}}$, the aircraft’s speed vector expressed in the ECEF frame, may then be calculated through successive rotations of the $v_{\text{NED}}$ vector about the Latitude and Longitude angles, also using quaternion algebra, as described in [12]:

$$v_{\text{ECEF}} = (q_\varphi^* q_\lambda) v_{\text{NED}} (q_\varphi q_\lambda)$$

where $q_\varphi$ and $q_\lambda$ are the appropriate rotation quaternions – dependent on the local Latitude and Longitude – to rotate the NED frame into the ECEF frame, and $q_\varphi^*$ and $q_\lambda^*$ are their complex conjugates.

The assumption of leveled flight (and thus zero vertical speed) means that $v_{\text{ECEF}}$ is a tangential speed in an Earth spherical coordinates system, and the aircraft’s angular velocity may be obtained using $v_{\text{ECEF}}$ and $r$, the aircraft’s position vector in ECEF coordinates, rearranging the tangential speed expression:

$$v_{\text{ECEF}} = r \times \omega \Rightarrow \omega = \frac{r \times v_{\text{ECEF}}}{||r||^2}$$

As shown in [16], a quaternion representing a rotation of $\omega$ over a $\Delta t$ time interval (in our case, the algorithm time step) may be calculated as:

$$q_{\text{ROT}} = \left( \frac{\omega \Delta t}{||\omega||^2} \sin \frac{\alpha}{2} \right)$$

$$\begin{bmatrix} \cos \frac{\alpha}{2} \\ \frac{\omega x}{||\omega||} \sin \frac{\alpha}{2} \\ \frac{\omega y}{||\omega||} \sin \frac{\alpha}{2} \\ \frac{\omega z}{||\omega||} \sin \frac{\alpha}{2} \end{bmatrix}$$

where $\alpha = ||\omega|| \Delta t$ is the angle of rotation about $\omega$.

Finally, the algorithm can compute the position quaternion at iteration $k + 1$ by simply applying quaternion multiplication between the computed rotation quaternion and the current position quaternion:

$$q_P[k + 1] = q_{\text{ROT}} \otimes q_P[k]$$

$q_P[k + 1]$ specifies a unique aircraft position, and may be transformed into geographical coordinates in the next iteration.

### 2.3 Uncertainty modeling

A real ATC environment is highly non-deterministic, as many factors are unpredictable. Common disturbances affecting trajectory prediction include: aircrew reaction time, aircraft performance variation, delay in instruction issuing due to a controller’s high workload or to a
temporary HF communication degradation, wind, unforecasted weather phenomenons.

Enroute oceanic operation outside radar coverage is a particular case in which the lack of information is dealt with by executing highly strategical planning (rather than tactical) ahead of time and by increasing dramatically the horizontal separation criteria. Therefore, short time scale disturbances affecting maneuver execution time are not critical, and may be considered inexistent. Wind, however, has to be accounted for: since aircrafts fly at constant airspeed, strong winds along the aircraft’s longitudinal direction may have considerable effect on ground speed, and thus alter the estimated time of arrival (ETA) at waypoints. Although the use of wind charts mitigates the error and allows the calculation of reasonably accurate ETA at waypoints, wind was still introduced as an uncertainty factor in the developed simulator.

3 Conflict Detection and Resolution

At the solution-search phase of the algorithm, aircraft trajectories are calculated deterministically. Conflict detection is carried out by checking horizontal and vertical separation minima used in ATC by each pair of aircrafts in the scenario. If a pair of aircrafts violates both separation criteria simultaneously at some time within the simulation window, the plan being tested is rejected, and an alternative solution has to be found.

If a search conducted in this manner generates a conflict-free plan, a robustness check is then carried out: a Monte Carlo simulation is run, in which several particles are created for each flight and different trajectories are calculated by adding a randomly generated wind vector to each particle. Without loss of generality, wind is modelled assuming a gaussian distribution. The conflict-free calculated plan is applied to every particle independently. In this case, each flight has a stochastic trajectory, and a pairwise conflict detection routine cannot return a binary output (conflict existence/absence) but is rather based on the concept of conflict probability.

At a given time, the algorithm runs pairwise conflict detection. Given two aircrafts $i$ and $j$, the number of conflicts $N_c$ may be found by testing every possible pair of particles from $i$ and $j$. Let $N_p$ be the number of particles for each aircraft, the number of combinations is $N_p^2$. Then, similarly to what is done in [17], the conflict probability between aircrafts $i$ and $j$ is estimated by the formula:

$$\hat{P}(C_{ij}) = \frac{N_c}{N_p^2}$$

A potential conflict is assumed to exist between $i$ and $j$ if $P(C_{ij})$ exceeds a threshold probability $p_{thr}$. In that case, the plan is rejected.

Conflict probability calculated in the above manner has a resolution equal to $\frac{1}{N_p^2}$, so the number of particles $N_p$ must be increased in order to obtain lower resolutions and to be able to detect scenarios with lower conflict probability.

3.1 Cost Function

Several different criteria may be used to evaluate a plan issued to an individual flight – trajectory change relative to FPL, number of instructions issued, total time inside FIR, fuel consumption – and there are also criteria suited to qualify a global plan – how much certain flights are penalized relative to others, controller’s workload peak at a given period. In this work, only two criteria are used in the Cost Function: (1) vertical deviation from the filed FPL (deviation cost $D$) and (2) total amount of ATC instructions (number of instructions $N$).

Since no horizontal instructions are issued and aircrafts fly their nominal horizontal route, the trajectories will only depart vertically from those requested by the crew, and the trajectory deviation from FPL will be calculated as an integral between filed FL (in the FPL) and the instructed FL (by ATC) along the path. Let $[WP_1,\ldots, WP_{N_w}]$ be a FPL route containing $N_w$ waypoints. For an aircraft $j$ flying between waypoints $WP_i$ and $WP_{i+1}$, the deviation integral is:

$$I_{j,WP_i-WP_{i+1}} = \int_{WP_i}^{WP_{i+1}} (FL_{FPL} - FL_{ATC}) ds$$

The deviation cost $D$ for each route segment of a flight depends not only on its deviation integral $I$, but also on whether the aircraft is flying above or below its requested FPL altitude, and whether it is
in its climb or descent phase. Figure 2 summarizes how deviation cost $D$ is calculated as a function of deviation integral $I$.

**Figure 2:** Deviation cost $D$ calculation depending on deviation sign and flight phase

Assuming flight $j$ has a FPL route containing $N_w$ waypoints, its total deviation cost is the sum of the deviation costs for each segment of its route:

$$D_j = \sum_{n=1}^{N_w-1} D_n$$

where $D_n$ is the deviation cost for route segment $n$. The total deviation cost for a global plan solving a scenario with $N_f$ flights is the sum of every individual flight deviation costs:

$$D_{total} = \sum_{j=1}^{N_f} D_j$$

Finally, assuming that plan is composed of $N_I$ instructions, a linear cost function may be defined as follows:

$$f = \lambda_N N_I + \lambda_D D_{total}$$

where $\lambda_N$ and $\lambda_D$ are weight coefficients. These parameters may be adjusted to favor one of the criteria over the other. Setting $\lambda_N < \lambda_D$ will cause the algorithm to instruct aircrafts to fly as closely to their FPL as traffic allows, no matter how many instructions it takes, whereas letting $\lambda_N > \lambda_D$ will favor the choice of plans with as few instructions as possible, disregarding vertical deviation from the requested flight levels.

### 3.2 Combinatorial Optimization

Some assumptions have to be made in order to define the solution search algorithm:

- The algorithm only acts upon the aircraft trajectories vertically, by issuing flight level instructions only. This means every aircraft flies its FPL route, corresponding to Great Circle paths between route waypoints, at the requested True Airspeed or Mach number. This is usually the case for real oceanic en route control, as vertical separation is a safe way to ensure safety in the absence of radar image.

- Time is discretized by only allowing instructions to be issued to an aircraft as it crosses a waypoint on its route, and not at some point in the middle of a route segment. This design option makes the state space dimension manageable for most traffic scenarios and is a logical choice, considering that pilots are required to contact ATC to report their position whenever they cross waypoints. This way, the controller (and the algorithm as well) knows when each vertical maneuver begins.

- Altitude is discrete by the very nature of the flight rules used in ATC, as aircrafts crossing upper airspace are required to cruise at flight levels separated by the standard vertical separation 1000 ft. This yields a finite set of altitudes that the algorithm can assign each aircraft to, bounded from above by the aircraft model’s flight ceiling.

The algorithm runs a branch-and-bound method to reduce computation time, assuring optimality. Starting at Current Time (the real time being observed by the human controller), the search advances progressively through time, generating predicted trajectories for each aircraft. Every time an aircraft reaches a waypoint, a node is created and several branches are considered, corresponding to all possible flight levels that aircraft can be assigned to. A lower bound of the cost is computed for each branch, measuring the minimum cost each one will add to the total cost function. The branch with the lowest heuristic is chosen, the corresponding action is executed upon the aircraft under analysis, and the search proceeds, until some aircraft reaches its next waypoint.

At each node, conflict detection is executed in the time window between the previous node and the current one. If a conflict between a pair of aircrafts $A$ and $B$ is detected, the algorithm backjumps to
the last node (the most advanced in time) where a decision was made involving either A or B, discarding every node in between, and selects a different branch to proceed the search.

A solution plan is found when the simulation time window limit is reached, or when every flight in the scenario has already abandoned the airspace being controlled. After the first solution plan has been found, an upper bound is available and may be used to prune branches in the middle of the search tree, whenever a node has a cost higher than the current upper bound. This greatly reduces the run time of the algorithm, requiring the expansion of fewer nodes.

The simulation is concluded when every node in the search tree has been considered, either by being expanded or by being pruned. Following the branch-and-bound method, branches are pruned whenever their lower bound exceeds the current upper bound \( (i.e., \text{the minimum cost among all solutions found so far}) \). This plan is selected to be presented as a suggestion to the controller.

In order to further reduce the computation time, the problem is decomposed using a graph representation. Each simulation scenario is analysed as an undirected graph, where each flight corresponds to a node. An edge is created between two vertices if the trajectories of the two corresponding flights are separated by less than the horizontal separation minimum at any pair of points, regardless of the time at which each flight crosses that point. Consider now this graph’s connected components (maximal sets of nodes connected among them by at least one path). Each connected component corresponds to a set of aircrafts where a conflict may occur. However, if any two aircrafts belong to different connected components, there is no possibility of conflict. Thus, the set of aircraft belonging to each connected component can be analysed separately. Having subdivided the aircraft set into connected components, the algorithm runs independent search trees for each, greatly reducing the amount of expanded nodes and the time needed to find an optimal solution.

### 3.3 Interactive decision making

The optimization algorithm described above assumes a fixed value \( \frac{\lambda_N}{\lambda_D} \), which in practice defines an ‘exchange rate’ between number of instructions \( N \) and deviation \( D \). This ratio is by nature arbitrary if the compared criteria are not directly comparable, as is the case. Hence, a multi-criteria problem is defined, and must be resolved by different methods than the ones used for conventional single-variable optimization. The goal is no longer to find an \textit{optimal solution}, but rather the set of \textit{Pareto optimal solutions}, also referred to as the \textit{Pareto front}.

In this work, the strategy employed to calculate the Pareto front is interactive. An interactive method avoids the (possibly time consuming) calculation of the whole Pareto front, and uses decision maker’s preferences to guide the algorithm towards the preferred solution. This means an interactive method inserts the decision maker in the optimization loop. The algorithm informs the controller each time a Pareto optimal solution is found, and the controller may choose to accept it or request a recalculation, introducing restrictions in \( N \) and/or \( D \). The simulation ends when the controller accepts one of the proposed solutions or when the whole Pareto front has already been calculated.

Figure 3 illustrates one of the advantages of this method. The approach known as \textit{weighted sum}, which consists of finding the Pareto optimal solutions simply by varying the criteria weight ratio \( \frac{\lambda_N}{\lambda_D} \) (equivalent to defining lines of varying slope, as the lines denoted \( m_1 \) to \( m_{10} \) in the figure), presents the drawback of being unable to find efficient solutions which are \textit{non-convex}. A non-convex efficient solution is one that, although being non-dominated, is not the optimal solution for any weight ratio \( \frac{\lambda_N}{\lambda_D} \). One example of a non-convex efficient solution is \( S_3 \), and the algorithm implemented in this work is able to find it, provided the appropriate restrictions are introduced by the controller.

### 4 Simulation Results

In order to evaluate empirically the approach, a number of simulations were ran, under the following assumptions:

- The controlled area consisted of Santa Maria (LPPO), the oceanic area under Portuguese jurisdiction.
- A time window \( (T_{\text{sim}}) \) of 3 hours was used.
- The dynamical system was integrated using \( T = 15 \text{ s} \).
• Distances of 1000 ft and 40 nautical miles were used for the vertical and horizontal separation minima.

• All flights were simulated using an Airbus 320 model, using BADA data.

In this section, an ATC instruction clearing a flight with callsign $c$ to change its cruising level to flight level $l$ at waypoint $w$ is represented as $I(c, w, l)$.

4.1 Algorithm performance

In order to evaluate the performance of the implemented algorithm, tests were ran with randomly generated scenarios of varying traffic densities (defined by the number of flights $NF$), and the values for running time and memory usage were observed.

Several simulations with randomly generated scenarios were run for each value of $NF$, measuring running time $t_{run}$. $NF$ was progressively increased, and mean values of $t_{run}$ were calculated. The evolution of $t_{run}$ as a function of $NF$ is shown in figure 4.

The obtained graph evidences an approximately exponential growth of running time as the number of flights increases, starting at about $NF = 25$. This clearly indicates the complexity of a given scenario – i.e. the computational effort it requires from the algorithm – grows much faster than the state-space dimension, which increases linearly with the number of flights $NF$. The explanation of this growth is rather intricate and somehow outside the scope of this work (as it deals with the factors affecting traffic complexity, such as the relative geometry of the aircraft’s trajectories) but it may attributed to the combinatorial effect as more aircraft are added to a scenario, increasing the number of conflicting aircraft, and forcing the algorithm to explore a much larger percentage of the search tree.

The mean values for the maximum memory usage during a simulation for increasing $NF$ are plotted in figure 5.

The graph in figure 5 clearly shows a linear variation of memory usage with increasing $NF$. This was to be expected, as the linear increase in the number of flights increases linearly the memory occupied by saved trajectory data, which comprises the largest part of stored memory. Since the algorithm stores just one set of trajectories and one search tree path at a time, no memory increment occurs as a consequence of a largest percentage of the search tree being explored, hence no exponential behavior is observed, contrary to the case of running time.
4.2 Optimality analysis

In order to quantify the effect of seeking optimality in the total costs of the obtained solutions, an optimality gain $G_{\text{opt}}$ is defined. This quantity is calculated as:

$$G_{\text{opt}} = \frac{f_{\text{greedy}} - f_{\text{optimal}}}{f_{\text{greedy}}} \times 100\% [11]$$

where $f_{\text{optimal}}$ is the cost of the optimal solution and $f_{\text{greedy}}$ is the cost of the greedy solution for a given simulation.

The optimality gain $G_{\text{opt}}$ is plotted as a function of $NF$ in figure 6.

![Figure 6: Optimality gain $G_{\text{opt}}$ variation with $NF$](image)

A first analysis of the figure reveals the optimality gain increases with the increase in the number of flights. This happens due to the occurrence of complex scenarios becoming more common when $NF$ is increased, which causes the difference between greedy and optimal solutions to be more evident. In other words, it may be said that the more complex the scenario, the larger the ‘room for improvement’ that exists for an optimal algorithm.

Secondly, it is worth observing that the optimality gain $G_{\text{opt}}$ exhibits a high variability, which is itself a consequence of the high variability of traffic complexity. Even for a high traffic density (high $NF$), many scenarios are created that are not complex enough to cause a considerable difference between the greedy solution and the optimal solution, which is indicated by the median values. However, there is a considerable percentage of randomly generated scenarios (and, presumably, of real operational scenarios) that thoroughly justify the calculation of an optimal solution, and these are the ones that underline the utility of the developed algorithm.

5 Conclusions

Motivated by the increase in air traffic and by the subsequent capacity overload verified for some airspaces, this work follows the approach of developing tools to assist air traffic controllers and enhance their performance.

A method is presented to execute the tasks of detecting and resolving conflicts automatically, by predicting the future trajectories of the aircraft present at the controlled airspace and by finding a plan that simultaneously ensures a safe separation for all flights and minimizes a defined cost function, which depends on the issued number of instructions and on the deviation from flights from their requested flight plans. In order to find this optimal solution for a given scenario, a branch-and-bound algorithm was implemented. This algorithm is both optimal and complete.

The developed algorithm was subjected to an extensive set of simulations, with varying levels of traffic density. Traffic generation was randomized to obtain a high variety of traffic patterns and geometries, hence varying traffic complexity. The algorithm was analyzed regarding its computational performance, exhibiting an exponential increase in running time as traffic density increases but only a linear increase in memory usage.

Regarding the cost of the calculated solutions, an improvement was verified between the greedy solution – used as a benchmark – and the optimal solution for each simulation, especially as traffic density is increased. Although great variability exists in traffic complexity, and thus in the improvement brought on by finding the optimal solution instead of settling for the greedy one, the developed tool excelled and proved its utility for a non-neglectable percentage of the randomly generated traffic scenarios, by achieving a high reduction in plan cost.

No attempt was made in this work to quantify ATCO workload time reduction and aircraft’s fuel savings, as such calculations fall somewhat outside the scope of this thesis and would be highly speculative, but it is safe to affirm that the application of the proposed method in operational context could be beneficial for both controllers and operators. At the very least, and even if the scenario complexity does not cause the optimal solution to present a considerable cost reduction relative to a greedy solution, the tool works as yet another mechanism
to ensure the safe separation between aircraft and as a means to reduce the time spent by controllers separating traffic, freeing them to perform other tasks.

References