



Transport and Telecommunication, 2023, volume 24, no. 3, 266–277
Transport and Telecommunication Institute, Lomonosova 1, Riga, LV-1019, Latvia
DOI 10.2478/ttj-2023-0021

AIR TRAFFIC MANAGEMENT USING A GPU-ACCELERATED GENETIC ALGORITHM

**Rahul Rampure, Raghav Tiruvallur, Vybhav Acharya,
Shashank Navad, Preethi P**

*Department of Computer Science and Engineering, PES University
100 Feet Ring Road, Bangalore, India, 560085
rahulrampur@hotmai.com, rtiruvallur@gmail.com, vybhavacharya@yahoo.com,
shashanknavad@gmail.com, preethip@pes.edu*

Air traffic management is becoming highly complex with the rapid increase in the number of commercial and cargo flights, leading to increased traffic congestion and flight delays. To mitigate these issues, we present a flight path generation system that distributes the aeroplanes across the airspace and imparts minimal delays to the flight if required, thus ensuring that the aircraft follows the shortest route wherein it encounters the least amount of traffic. We develop a parallel genetic algorithm in CUDA-C with a novel fitness function allowing the system to reach an optimal solution where the air traffic density is minimised. The proposed algorithm was tested on one day's domestic flight schedule and achieved an 18% reduction in traffic density, with the flight times and delays remaining proportional to the data observed in the existing air traffic management system.

Keywords: Genetic Algorithm, Air Traffic Management, CUDA, Spectral Clustering

1. Introduction

Air traffic management is a flow management problem wherein multiple aircraft traverse through a shared network and was first explored in (Odoni, 1987). The basic formulation of the problem was the division of the airspace into a set of geometric shapes called 'sectors,' with these sectors forming an Air Traffic Control (ATC) network. Airports were present within these sectors and were considered the source and sink for the commodity being transported, which is the aircraft. Each sector has a 'capacity' defined as the maximum number of aircraft present simultaneously within it. To ensure that capacity constraints are met in all sectors, the aeroplanes should be evenly distributed across the network, preventing any sector from becoming a traffic hotspot.

The solution to this flow management problem was explored in various works such as (Sun *et al.*, 2011), where a cell transmission model was proposed to represent the problem, and a solution was obtained using an integer program that was then decomposed into a linear program. A hierarchical planning framework was proposed in (Zhang *et al.*, 2011) that presented a planning algorithm that considered weather impact and aircraft types in the optimisation constraints. An integer program method was put forward in (Bertsimas and Patterson, 1998), which enforced a 'ground-holding' policy wherein flights are delayed at the departure airport, allowing for a more cost-effective flight than an airborne deviation due to traffic congestion. In all these works, the common goal was to develop a flow management system that satisfied the sector capacities, making the sector capacities a crucial input to the system, which may not always be known beforehand. Hence our work proposes a model that minimises these sector capacities themselves. An average drop of 18% is achieved compared to the actual sector capacities, reducing the number of traffic hotspots in the airspace.

The first step towards proposing a solution would be to sectorise the airspace to measure air traffic. The Federal Aviation Administration (FAA) has already proposed a network of predefined sectors, each referenced by a unique ID, and this network is part of the National Airspace System (NAS). This existing system and various sectorization methods are presented in (Basu *et al.*, 2009). Another method is proposed in (Vaidhun *et al.*, 2020), which divided the airspace into a set of hexagons, each associated with some potential energy. Our proposed method uses a spectral clustering algorithm (Ng *et al.*, 2001) to obtain the sectors. Upon forming a network of sectors, the next step would be the development of the genetic algorithm. We parallelise and improve upon the algorithm presented in (Ahn and Ramakrishna, 2002) by ensuring that instead of the shortest path, the fittest path is obtained, which optimises the fitness function

presented in Section 2.2. The usage of genetic algorithms for aerial path generation has been explored in multiple works, such as (Hu *et al.*, 2004, Zhenga *et al.*, 2004), which focused on unmanned aerial vehicles navigating a dynamic environment.

A similar work is presented in (Liu *et al.*, 2019), where particle swarm optimisation (PSO) is presented instead of a genetic algorithm. The main contribution of these works to the discussion was the development of a path fitness function that also considers other aerial vehicles. Our model was evaluated using the Aircraft Situation Display to Industry (ASDI) data for August 2013, which consisted of two data tables: the flight tracks and the flight history. The flight history data table represents the flight schedule which consists of the flight data for all the domestic flights that took place in the USA. The tracks table consists of a sequence of coordinates for each flight, representing its path. This data was obtained from the Flight Quest 2 competition organised by General Electric, which aimed to optimise flight path trajectories. A complete overview of this competition is found in (Kiss-Tóth and Takacs, 2014).

With the sectors defined, the navigation of each aeroplane within it, such that the system follows a no-collision policy, is handled by the Airborne Collision Avoidance System (ACAS) aboard each aircraft. The usage of collision avoidance systems, such as the reinforcement learning system presented in (Wen *et al.*, 2019), the ripple spreading algorithm presented in (Zhou and Hu, 2020), or the Kalman Filter method proposed in (He *et al.*, 2020), can further bolster it. Hence, a collision avoidance system is not implemented in our work because we assert that the possibility of collisions becomes minimal when the number of aircraft within each sector is minimised.

Our proposed method uses a genetic algorithm to distribute air traffic and minimise mid-air traffic. Two scenarios were considered when generating a flight path: either the aircraft takes a longer route with less traffic or delays its departure time and takes the shorter route when most of the traffic has cleared. The aim is to find the optimal balance between these two scenarios by weighing the amount of delay against the reduction in traffic. The shared airspace is modelled as a network of interconnected sectors, and a genetic algorithm implemented in CUDA is used to generate a flight path across this network. The fitness function used by the algorithm seeks to find the best approach, considering factors such as path length, traffic, and delay, to optimise the flight's efficiency.

This study also allocates runways to the departing and arriving aircraft at all airports, with the runway occupancy time (ROT) set as one minute. ROT is the interval of time an aircraft blocks a runway for take-off or landing, preventing its use by other aircraft. A complete overview of ROT is found in (Kumar *et al.*, 2009), which shows that the median ROT for most US airports is less than sixty seconds. With the busiest US airports having multiple runways, our algorithm maps each flight to a runway and delays the flight if all the runways are blocked. Hence, the maximum number of aircraft that can depart or arrive at an airport simultaneously in a minute is upper bound by the number of runways it has.

2. Proposed methodology

A genetic algorithm is an optimisation technique that belongs to the category of metaheuristic algorithms, like particle swarm optimisation, ant colony optimisation, tabu search, and many other methods that mimic biological processes. These algorithms use natural principles to find the optimal solution to an abstract problem or the minima or maxima of a multivariate function. The general idea would be to generate multiple candidate solutions to the problem, set up a method that evaluates their quality, and then establish a mechanism that allows them to coordinate with each other to generate a higher-quality solution until the best one is obtained. An exhaustive study about GAs can be found in (Goldberg, 1989).

A genetic algorithm mimics evolutionary mechanisms wherein a chromosome represents a candidate solution to the problem. A population is a collection of chromosomes or a set of candidate solutions, with the population size being one of the input parameters. The algorithm first generates an 'initial population' by randomly generating chromosomes and then performs five genetic operations: Selection, Crossover, Mutation, Repair, and Elimination, on the population in a loop, with each loop iteration called a generation. The loop terminates after the convergence condition is satisfied, which occurs when the best chromosome is unchanged for a set number of generations, and that would be the final solution returned. The problem we aim to address using the GA would be finding the best aerial path and the optimal ground delay for a single flight; hence, this would be the solution returned.

Each chromosome has a specific 'fitness' value which indicates its quality. The algorithm achieves the best solution to the problem by generating a chromosome with maximum fitness. The Selection operation selects the best chromosomes in the population based on their fitness values. The Crossover operation combines two chromosomes selected in the previous step to generate new, better chromosomes. The Repair operation is specific to our use case and improves chromosome fitness. Mutation enhances the

fitness of the final solution, and Elimination removes the worst set of chromosomes based on their fitness from the population.

Arriving at a solution using a genetic algorithm is inherently slow due to its iterative nature. The genetic operations are executed in a loop, and each operation iterates over the chromosomes in the population. Our proposed system has been evaluated using the flight schedule for a full day, which has thousands of flights. Hence, the genetic algorithm will be executed thousands of times, with each run generating the optimal path for one flight. To ensure that the algorithm was high performing and converged to a solution within a second, we parallelised the genetic operations in CUDA. We utilised the GPU, which has thousands of processing units, allowing for fine-grained parallelism.

Compute Unified Device Architecture (CUDA) is an API provided by Nvidia Corporation allowing us to defer the execution of the genetic operations to the GPU, which would be vastly beneficial compared to CPU execution, due to the vast number of threads that can be spawned. The CUDA model implements data parallelism by concurrently executing the same procedure using multiple threads but on different chromosomes, which is leveraged by the algorithm to avoid sequentially iterating over the chromosomes in the population.

For using the GA for flight path generation, the airspace is divided into a set of interconnected sectors and represented as a graph, and this procedure is depicted in section 2.1. The parallel genetic algorithm is detailed in section 2.2, which starts with a discussion on the representation of a flight path as a chromosome and the fitness function, followed by the genetic operations and their respective parallelisation strategies.

2.1. Airspace sectorization

This study used the ASDI Flight Tracks data table to obtain a month's flight tracks. A flight track is an aircraft's path during its flight, from its departure to arrival airport, represented as a sequence of coordinates measured at fixed time intervals. We obtain a set of two million unique coordinates, plotting them as blue dots in Figure 1(a).

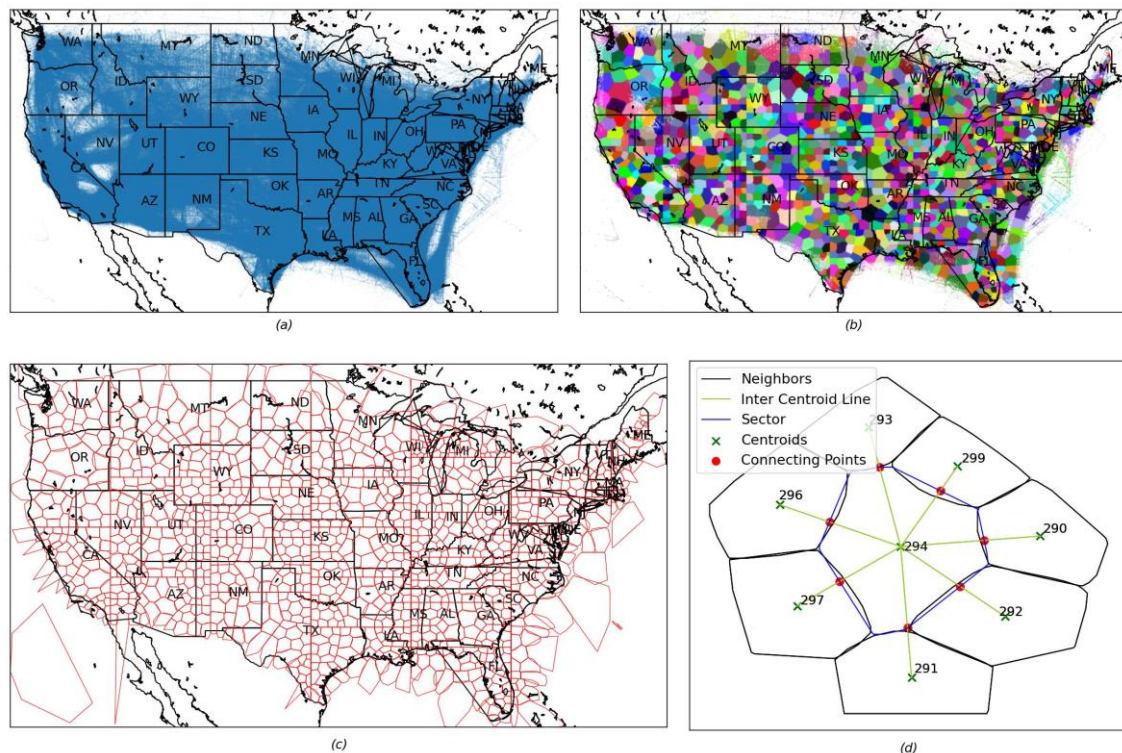


Figure 1. Tracks, Clusters, Sectors and Connecting Points

Spectral clustering, a density-based clustering algorithm, is used to form 1250 clusters of these coordinates. This operation is performed hierarchically to tackle the complexity associated with the algorithm, with one cluster being divided into subclusters. The final set of clusters is represented in

Figure 1(b), where each is depicted with a unique colour. Using Chan’s algorithm (Chan, 1996), a convex hull is formed out of each cluster. A convex hull is a set of points representing a closed figure if joined together. By constructing a convex hull out of each cluster, the airspace is divided into 1250 adjacent shapes, thus allowing us to define the airspace sectors as these hulls. The airspace sectors are depicted as red shapes in Figure 1(c).

With the sectors defined, a graph is created with sectors being the vertices and adjacent sectors being connected with an edge. Two sectors are adjacent if the minimum distance between them is less than ten thousand metres, forming 3725 edges and 1250 vertices. Each US airport is mapped to a sector based on its coordinates. A flight path is defined as a sequence of sectors across the graph, from the sector containing the departure airport to the sector containing the arrival airport.

Each sector is associated with a set of Connecting Points, as shown in Figure 1(d). These are the points or coordinates at which two adjacent sectors are connected. For example, in Figure 1(d), the connecting points for sector number 294 are shown in red. To obtain the connecting points for a sector, we first join its centroid with its neighbouring sectors’ centroids to form inter-centroid lines, then find the intersection of these inter-centroid lines with the hull/sector boundary and mark them as the connecting points. Thus, each sector is associated with its set of connecting points, which will be used for path fitness evaluation.

The advantage of our generated sectors over predefined NAS sectors is that the size and shape of the sectors become tuneable. In our work, we generated a greater number of sectors than the ones available in the NAS, since it allows for finer traffic distribution where the minimum inter-aircraft separation distance can be enforced as a restriction. The existing FAA sectors are fixed, and prevalent research in the aviation industry lays out their disadvantages, such as not reflecting real-time traffic and weather conditions, resulting in rigid flight routes. The NextGen air traffic control system, an ongoing research project by the FAA aimed at reducing traffic load, is closely associated with dynamic airspace sectorization, and a detailed discussion using spectral clustering is presented in (Li *et al.*, 2009).

2.2. Parallel genetic algorithm

The genetic algorithm is detailed in this section, demonstrating the genetic operations and the terminologies involved. Initially, we give a background on the representation of the chromosomes and their connection to the sectors created; this is followed by a discussion on the formation of the ‘fitness function,’ where the method for mathematically evaluating a flight path’s quality is established. This section is concluded with a description of the parallel genetic algorithm.

Any flight path in our model is represented by a sequence of sectors, which starts and ends with the sectors containing the departure and arrival airport, respectively. An example of a flight from Boise International Airport, Idaho (KBOI), to Billings Logan International Airport, Montana (KBIL) is illustrated in Figure 2. The flight follows a sequence of sectors, represented as an array, and denoted by S_p . The connecting points joined by a continuous line are depicted in the figure, where the two airports are the starting and final connecting points, representing the aircraft’s flight path. The Euclidean distance between each pair of consecutive points is computed and stored in an array denoted by D_p .

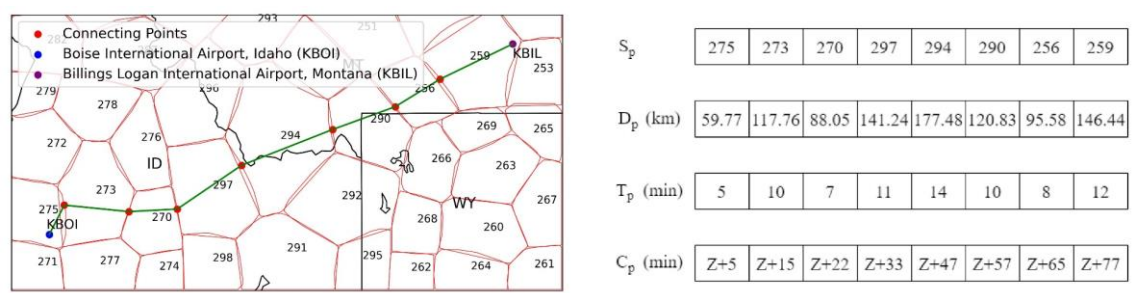


Figure 2. A Single Chromosome

The aircraft’s cruise ground speed is taken as an input to the system and is assumed to be constant throughout its flight. In this example, the cruise speed is taken as 400 knots and using the speed formula, the time spent in each sector of S_p is calculated and stored in an array denoted by T_p . Assuming the flight’s departure time is Z , another array (C_p) is defined, representing the time at which the aircraft exits each sector of S_p . We obtain this array by taking the cumulative sum of T_p and adding Z . Thus; each chromosome

consists of four arrays: S_p , D_p , T_p , and C_p . The above discussion is summarised using the example in Figure 2, which shows a single chromosome represented by these four zero-indexed arrays.

- $S_p[2]$ is the 2nd sector visited in the path, sector 270.
- $D_p[2]$ is the distance travelled in sector $S_p[2]$ or sector 270, equalling 88.05 km.
- $T_p[2]$ is the time spent in minutes in sector $S_p[2]$ or sector 270 and is equal to 7 min.
- $C_p[2]$ is the time at which the aircraft leaves $S_p[2]$ and enters $S_p[3]$, which implies the aircraft leaves sector 270 and enters sector 297 twenty-two minutes after its departure.

We also keep two matrices in addition to these data structures, the “Traffic Matrix,” or T_m , with 1250 rows and 2880 columns, and the “Airport Matrix,” or A_m , with 1103 rows and 2880 columns. Time is discretised into minutes, with 13:30 of the current day being equivalent to $13 \times 60 + 30 = 810$, and using this timing system, every minute in two days is mapped to a column of T_m and A_m .

Since the airspace was sectorised into 1250 sectors, each sector is mapped to a row of T_m , and $T_m[i][j]$ will be the number of aircraft present in the i^{th} sector in the j^{th} minute, which aids the algorithm in air traffic distribution. With 1103 airports in the USA, each one is mapped to a row of A_m , and $A_m[i][j]$ will be the number of runways in use in the i^{th} airport during the j^{th} minute, which enables the algorithm to enforce the ROT constraint discussed earlier.

Every flight has a scheduled departure time per the flight schedule, which our algorithm can delay, as mentioned previously, to find the best trade-off between the delay incurred and the reduction in mid-air traffic encountered. Thus, each chromosome has this delay factored into it in the C_p array, where Z represents the actual or the delayed departure time, which is obtained by adding the delay to the scheduled departure time. This delay may also be zero, which occurs when it is observed that adding a delay provides no benefit. Hence, each chromosome is associated with an optimal ground delay in addition to its fitness.

With the chromosome defined, we present the fitness function used in the algorithm for evaluating its quality, which is a numerical value denoting how desirable it is. There are multiple factors to consider within the fitness function, with contradictory relationships among them, and these factors are listed below. A combination of the values of these factors provides us with the final ‘fitness’ or quality value of a chromosome or a candidate flight path.

1. The path length (L_p) refers to the distance covered by an aircraft following the flight path represented by the chromosome, which equals the sum of the elements in the D_p array, as shown in (1). The path length must be minimised to ensure an economical flight.

$$L_p = \sum D_p \quad (1)$$

2. The path angular deviation (A_p) refers to the average turn taken by the aircraft while following the path represented by the chromosome. A_p is evaluated by joining the sequence of connecting points with lines and finding the deflection angle between each pair of adjacent lines, then taking the average of these deflection angles as shown (2). The deflection angles (θ) are illustrated in Figure 3(a) and must be minimised to prevent abrupt turns, which worsen the flying experience for the passengers. The average of all deflection angles was chosen because other methods such as maximum might add bias and would not be a fair assessment of the overall quality of the path.

$$A_p = \frac{\sum_{i=0}^n \theta_i}{n} \quad \text{where } n = \text{length}(S_p) - 2 \quad (2)$$

3. The traffic factor (T_f) measures how much mid-air traffic the aircraft encounters while traversing the path represented by the chromosome. T_m is used to calculate the traffic factor as shown in (3) and must be minimised.

$$T_f = \sum_{j=Z}^{j=C_p[0]} T_m[S_p[0]][j] + \sum_{i=1}^{n-1} \sum_{j=C_p[i-1]+1}^{C_p[i]} T_m[S_p[i]][j] \quad \text{where } n = \text{length}(S_p) \quad (3)$$

4. The ground delay (G_d) is the delay imparted to the aircraft at the departure airport to allow for take-off when there is reduced prospective air traffic. In Figure 3(b,c), we show the variations in T_f with G_d for two different chromosomes, and the general trend observed is that T_f reduces with the increase in G_d . The minima in the T_f vs. G_d plot are found, shown in the figure as blue points. The point with the least slope is selected, implying the least G_d for which the largest decrease in T_f is observed. The figure shows this point in red, and its x-coordinate is termed G_{od} , which is the optimal delay. The T_f observed at G_{od} is denoted by T_{of} . We mathematically represent this procedure for obtaining G_{od} in the (4).

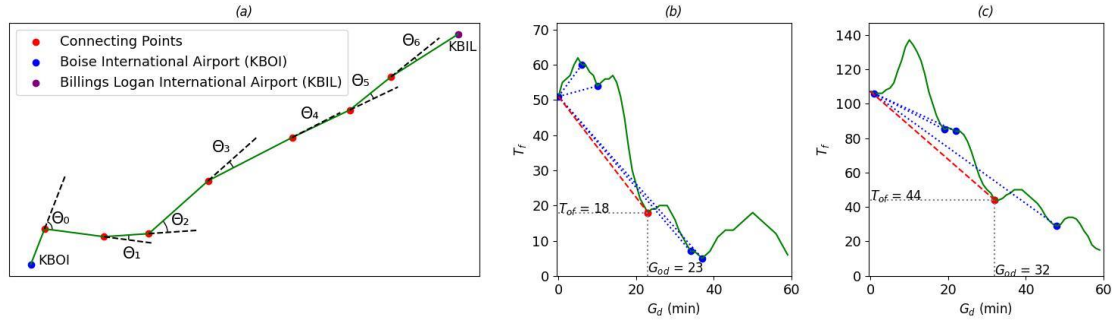


Figure 3. Deflection angles and optimal delay

$$G_{od} = \arg \min_{G_d} \left(\frac{\Delta T_f}{G_d} \right) \quad \text{s.t.} \quad \frac{dT_f}{dG_d} = 0 \text{ and } \frac{d^2 T_f}{dG_d^2} > 0. \quad (4)$$

The chromosome's fitness is set as zero if the number of runways being used in the same minute exceeds the number of runways available in either airport because it would mean that the aircraft uses a runway blocked by another aircraft, thus violating its ROT. This constraint is added to the GA, which is depicted in the final fitness function in (5). The GA must generate a chromosome that maximises the fitness function, which would be the optimal solution to be output. We represent the departure and arrival airports by A_d and A_a , respectively, and the number of runways in the departure and arrival airports by R_d and R_a , respectively. The Z value in the C_p array of the chromosome is set with the appropriate value using G_{od} . The sum of all the elements in T_m is represented as S , making $(S - T_{of})$ a positive quantity.

$$\text{Fitness}(\text{chromosome}) = \begin{cases} 0 & \text{if } A_m[A_d][Z] \geq R_d \text{ or } A_m[A_a][C_p[n-1]] \geq R_a \\ \frac{(180 - A_p) \times (S - T_{of})}{L_p} & \text{otherwise} \end{cases} \quad (5)$$

The path length (L_p) can become exceedingly large and hence it is placed as a standalone parameter in the denominator, as any other parameter would either be insignificant if added to it, or would skew the result towards it if multiplied. The path angular deviation (A_p) is measured in degrees and since it must be minimized, we include $(180 - A_p)$, a positive quantity, in the numerator. Similarly, $(S - T_{of})$ is also included in the numerator. Hence, when the algorithm maximizes the overall fitness value, it will minimize L_p , which is in the denominator, and maximize the two quantities $(180 - A_p)$ and $(S - T_{of})$, which are in the numerator, which in turn minimizes A_p and T_{of} . The two quantities in the numerator are multiplied with each other as illustrated in (5) rather than added since the amount $(S - T_{of})$ can grow extremely big and render the amount $(180 - A_p)$ insignificant.

The input provided to the algorithm is a flight schedule consisting of N flights, and for each of these flights, we have its A_d , A_a , scheduled departure time, and cruise speed (assumed to be constant), and the goal would be to generate a path for each. The genetic algorithm consists of five parallel operations executed in a loop until convergence, and it is run once for each flight in the input flight schedule. After each run, T_m and A_m are updated, so the system knows about the previously generated flight paths during its next run, using (6) and (7).

$$\begin{aligned} T_m[S_p[0]][j] &= T_m[S_p[0]][j] + 1 \quad \forall j \in [Z, C_p[0]] & \text{where } n = \text{length}(S_p) \\ T_m[S_p[i]][j] &= T_m[S_p[i]][j] + 1 \quad \forall i \in [1, n), j \in (C_p[i-1], C_p[i]] \end{aligned} \quad (6)$$

$$A_m[A_d][Z] = A_m[A_d][Z] + 1 \quad \text{and} \quad A_m[A_a][C_p[n-1]] = A_m[A_a][C_p[n-1]] + 1. \quad (7)$$

A high-level overview is shown in Figure 4, which shows the algorithm looping over the input flight schedule, generating a path for each flight. The algorithm starts by generating an initial population, which is a set of randomly generated flight paths upon which improvements are made. The five parallel operations involved, Selection, Crossover, Mutation, Repair, and Elimination, are executed in a loop until a

convergence condition is satisfied, upon which T_m and A_m are updated to reflect the path generated, and the algorithm moves on to the next flight in the schedule. We list all these processes below and their respective parallelisation strategies.

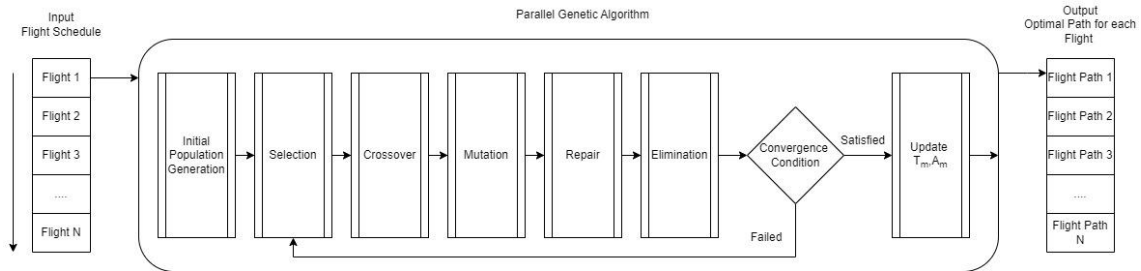


Figure 4. High Level Design of the Genetic Algorithm

Initial Population Generation

The algorithm first generates the initial population, a set of randomly generated flight paths or chromosomes, each associated with its fitness based on (5). This procedure allows us to obtain the population or the set of solutions upon which the genetic operations are applied. A large population size would result in the algorithm converging to a high-quality solution. However, as sequentially generating a large number of chromosomes is time-consuming, they are parallelly generated using the GPU.

We obtain the sectors where the departure and arrival airports are present and run a randomised variant of the depth-first search (DFS) technique on the graph between them. The DFS works by randomly picking neighbouring vertices continuously to form a path across the graph until the arrival airport's sector is reached. If a dead end is encountered where every neighbouring vertex has been visited, the DFS is restarted, reinitialising the chromosome. This contrasts with a simple DFS in that we do not pick the first neighbour at a vertex and do not backtrack at a dead end. This type of randomised DFS was found to be more time-efficient when searching in the grid-type structure of the airspace sectors compared to a traditional DFS. This process produces the S_p array of the chromosome, which is used to construct the other three arrays, namely D_p , T_p , and C_p , of the chromosome and is then linked with its fitness value using (5). It must be noted that the fitness function may change the C_p array of the chromosome by modifying the departure time Z by adding G_{od} . The initial population of size P is generated by spawning P threads on the GPU. Each thread generates a chromosome parallelly by performing its own randomised DFS. Each thread also calls the fitness function parallelly; hence, the fitness function is implemented as a device kernel to be called by only the CUDA threads.

Selection

The selection operation chooses chromosomes with a high fitness value to be combined in the crossover operation. This ensures that the algorithm always propagates good chromosomes allowing the population to be driven to the best solution. All the chromosomes in the population are randomly grouped into pairs of two, forming $P/2$ such pairs, where P is the population size. The fitness values are compared for each pair of chromosomes, and the one with the higher fitness value is selected. Since one chromosome is selected from each pair, the number of selected chromosomes is $P/2$. We spawn $P/2$ threads, one for each chromosome pair, and each thread selects one chromosome in parallel.

Crossover

The crossover operation involves combining two chromosomes selected in the previous step to produce new chromosomes that may have a higher fitness value. This operation allows the members of the population to coordinate with each other and try to generate a chromosome that maximises the fitness function's value. The selected chromosomes are randomly grouped into pairs of two, forming $P/4$ such pairs since $P/2$ chromosomes were selected in the previous step. Each pair forms the parent chromosomes combined to form a pair of offspring chromosomes. On the GPU, we spawn $P/4$ threads, each of which crosses over a pair of chromosomes to produce two offspring chromosomes. Hence $P/2$ offspring chromosomes are produced and added to the population, making the population size $3P/2$. The crossover operation performed by each thread in parallel is described using an example below.

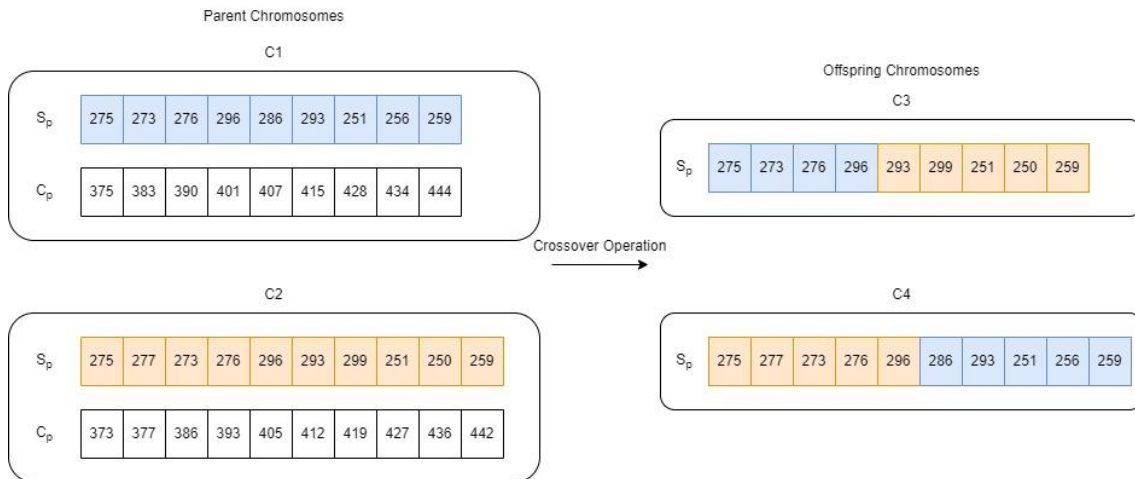


Figure 5. Crossover Operation

We use a variant of the single-point crossover method that accounts for both flight delays and the position of an aircraft that changes over time. The basic idea is shown in Figure 5, which has two parent chromosomes, C1 and C2, interchange parts of their S_p arrays to obtain the S_p arrays of C3 and C4, the pair of offspring chromosomes. Their remaining arrays, namely D_p , T_p , and C_p , are evaluated after the crossover process is finished, along with their fitness value. The point at which this interchange occurs is determined using the C_p arrays of the two parent chromosomes and is called the crossover point, with the crossover point being 296 in the figure. For choosing such crossover points, we find sectors traversed in a common time interval in the parent chromosomes. For example, in Figure 5, we say that 296 is a crossover point because the aircraft crosses it in the time interval (390,401] in C1 and correspondingly in (393,405] in C2, making the time interval [393, 401] common between the two. Similarly, 273 and 293 are also crossover points; thus, multiple such crossover points can exist, and we randomly choose one to initiate the interchange process.

Mutation

The mutation operation entails randomly choosing a set of chromosomes to be modified every generation or iteration of the genetic algorithm. This procedure increases the fitness value of the final chromosome output by the algorithm. The operation involves choosing a random sector in the S_p array of the chromosome, starting a randomised DFS from that sector to the arrival airport's sector, and then overwriting the corresponding part of that chromosome's S_p array. We have already detailed the randomised DFS procedure for generating the initial population. The mutated chromosome replaces the original in the population, and its D_p , T_p , and C_p arrays, along with its fitness value, are recalculated. The number of chromosomes to be mutated per generation, M , is taken as an input. We spawn M threads, and each thread concurrently mutates one randomly selected chromosome from the population.

Repair

Since cycles would increase the path length, the chromosomes or paths must be free from any cycles to have a high fitness value. In crossover and mutation, there is a chance that cycles were introduced in the S_p arrays of the chromosomes. The repair operation maintains a hash table storing the latest index of each sector in the S_p array of the chromosome. Any repeated sector, which indicates a cycle, is eliminated as the previous index of the repeated sector gets overwritten. The repaired path without cycles can be obtained by scanning the hash table and arranging the sectors in the increasing order of their index. This path replaces the original in the population, and its fitness value is re-evaluated, along with its D_p , T_p , and C_p arrays. We spawn $3P/2$ threads on the GPU, one for each chromosome in the population, and each thread will perform the repair operation on its allocated chromosome in parallel. The chromosome will be left as it is if no cycle is detected.

Elimination

The elimination operation removes chromosomes with a very low fitness value from the population, allowing the algorithm to converge to an optimal solution within a few generations. With the population

size being $3P/2$, the elimination operation removes the worst $P/2$ chromosomes, bringing the population size back to P . It does so by sorting the population based on the increasing fitness values of the chromosomes and then discarding the worst $P/2$ chromosomes. Thus, the elimination process serves two purposes: to keep the population size constant at P and to work with the selection operation to drive the population to the optimal solution quickly. Without the elimination operation, it was observed that the algorithm converged to a solution after many generations, increasing the execution time of the algorithm. We utilised the CUDA thrust library, which provides a GPU-accelerated parallel quicksort function to perform this operation.

Convergence condition

The convergence condition checks if the population has converged to a solution or if the algorithm can exit from the loop of genetic operations. The algorithm moves on to the next loop iteration if this condition is not satisfied. The condition checked is whether the chromosome with the highest fitness value is unchanged for 20 generations, which indicates that the algorithm could not find a chromosome with a fitness value that surpasses the best one in the population for 20 consecutive loop iterations. If this condition is satisfied, there would be no reason to continue the loop, and the chromosome with the highest fitness value is output as the final solution.

3. Results and Discussion

To evaluate how well the proposed model performs compared to the existing air traffic management system the FAA uses, we contrast the flight paths generated by our model with the flight paths taken in reality. We use the ASDI flight history data table to obtain the domestic flight schedule for August 19, 2013. This list contains each flight's scheduled departure time, flight time, and delay. The time the aircraft spends in the air is denoted as its flight time, and flight delay is the time by which its departure gets delayed at the departure airport. The algorithm is executed once for each of the roughly twenty - four thousand flights in the schedule, and each run generates a flight's optimal path and predicts its flight duration and delay. The machine used for testing had an Nvidia RTX 3050 GPU, an AMD Ryzen 7 CPU, and 16GB of RAM; the execution time was around four hours.

The proposed model's primary objective is to distribute the traffic and reduce traffic congestion while ensuring that the flight times and delays remain proportional to what is observed in the existing model. Air traffic congestion is measured using the terms "Sector Density" and "Sector Occupancy," which signify the maximum and the average number of aircraft present simultaneously in a sector. The solution generated by the proposed model yields an array of 1250 sector densities and occupancies, as well as the predicted flight times and delays for each flight in the input schedule. We use the flight tracks data to get the corresponding arrays of sector densities and occupancies, along with the flight times and delays in the existing model.

In Figure 6(a), we show a histogram with the x-axis representing real flight time – predicted flight time, and it is observed that the peak occurs at around 10 minutes, with the average difference being 6.5 minutes. This variation is attributed to the assumption that the aircraft maintains a constant cruise speed throughout its flight without accounting for the acceleration and deceleration at take-off and landing, respectively. Figures 6(b) and 6(c) compare the flight delays in the existing and proposed models using two histograms, respectively, and we show their averages in Table 2. The delays are similar, except for many outliers where the flight delays in the existing model were more than an hour, which we attribute to delays due to factors unrelated to traffic, such as inclement weather. Figure 6(d) depicts a bar plot showing the sector densities on the x-axis. We observe a higher number of sectors having less sector density in the proposed model than the existing one, directly implying the reduction of traffic congestion and elimination of hotspots. We note that the average sector density was reduced by 18%, and the average sector occupancy was reduced by 9%, as shown in Table 1.

Figure 6(e) shows a part of the traffic heatmap generated from the existing model, where a gradient is used to represent sector densities. A disproportionate distribution of air traffic is observed due to the uneven shading of the sectors, with some sectors being close to white and others being close to black, signifying the existence of hotspots or sectors with traffic congestion. The traffic heatmap from the proposed model is represented in Figure 6(f), which depicts the uniform distribution of air traffic due to the sectors being shaded evenly with grey. A simulator was developed in python and used to observe the distribution of air traffic, which can be observed in Figure 7, which shows the aircraft using coloured dots on the sector map. Figure 7(a) shows the state of the system at time 4:50. Figure 7(b) shows the state of the

system 20 minutes later at 5:10. Mid-air separation between the aeroplanes can be seen in 7(a) and one also observe that the aircraft form multiple queues as they approach the arrival airport in 7(b).

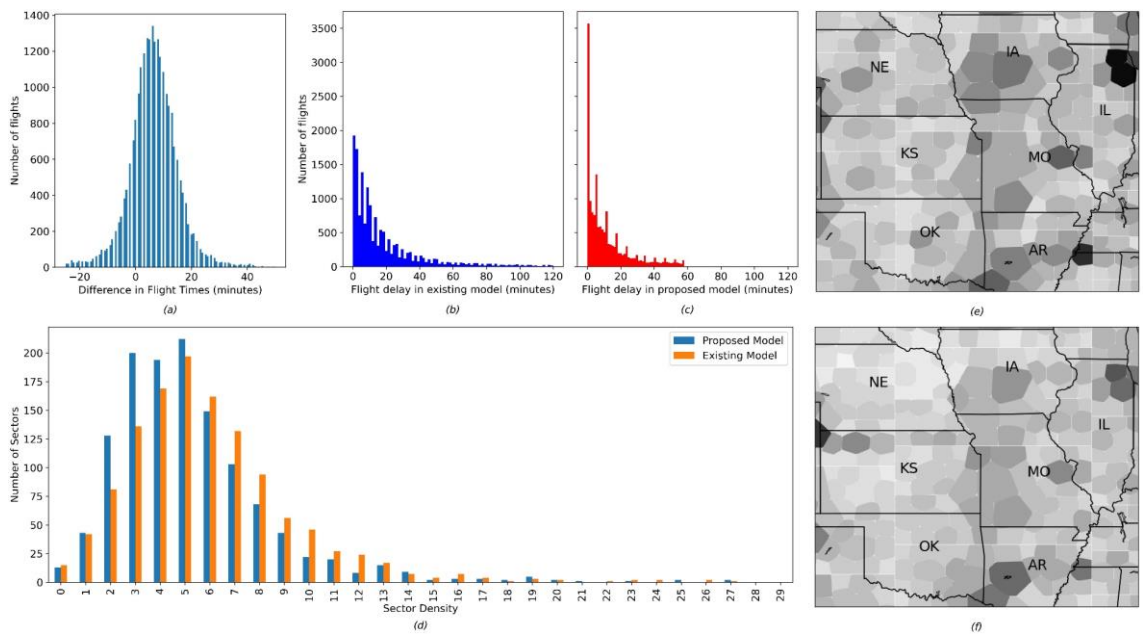


Figure 6. Analysis of the metrics and the traffic heatmaps

Table 1. Traffic Reduction

	Existing Model	Proposed Model	Reduction	Percentage reduction
Average Sector Density	6.42	5.26	1.16	18.06%
Average Sector Occupancy	2.05	1.87	0.18	8.78%

Table 2. Flight Delays

	Existing Model	Proposed Model	Reduction
Average Flight Delay	18.26 min	13.14 min	5.12 min

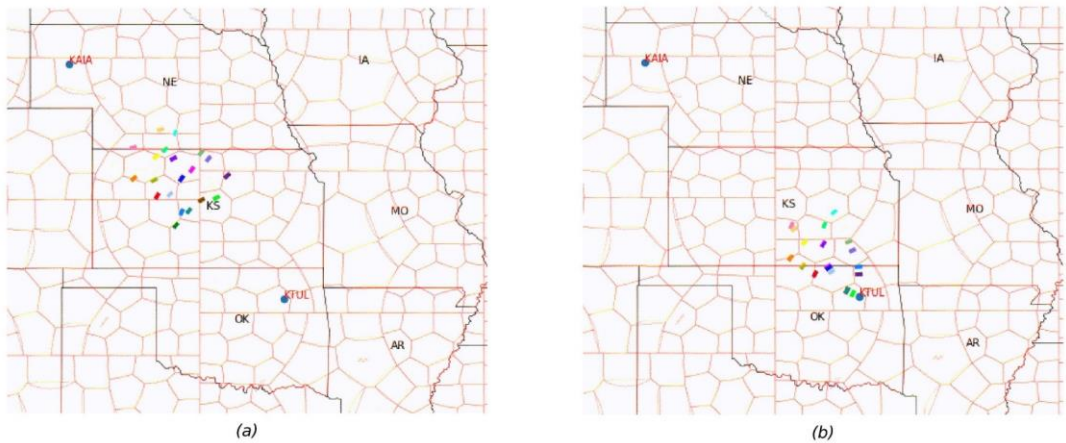


Figure 7. Distributed air traffic observed from our proposed algorithm

4. Conclusion

In this study, we have presented a novel technique to control air traffic by developing a parallel genetic algorithm that uses the GPU to produce flight paths, ensuring that an aircraft follows the path where it encounters the least traffic. The sectors were generated using spectral clustering on the points in the flight tracks data, and a unique method was proposed to measure air traffic in the algorithm quantitatively. We developed a fitness function that reduces the amount of traffic encountered, the path length, the path angle deviation, and the amount of delays introduced. The various operations involved in the algorithm and the method employed to parallelise each is demonstrated. Compared to existing air traffic management models used by the FAA in the USA, our methodology ensures that aircraft delays are minimal and that flight times remain the same while reducing airspace traffic congestion by an average of 18%. By including the weather and a flight path's aerodynamic effectiveness, we hope to further enhance the model in the future. The GitHub repository for the work presented here is located at (<https://github.com/CascadingRadium/Air-Traffic-Distribution>), which has the CUDA code for the algorithm and the simulator, along with a web-based user interface to allow the user to input the flight schedule, execute the algorithm, obtain the result, and simulate it easily.

References

1. Ahn, C. W., and Ramakrishna, R. S. (2002) A genetic algorithm for shortest path routing problem and the sizing of populations. *IEEE transactions on evolutionary computation*, 6(6), 566-579. DOI: 10.1109/TEVC.2002.804323.
2. Basu, A., Mitchell, J.S.B., Sabhnani, G.K. (2009) Geometric algorithms for optimal airspace design and air traffic controller workload balancing. *Journal of Experimental Algorithmics (JEA)*, 14. DOI: 10.1145/1498698.1537598.
3. Bertsimas, D., and Patterson, S. S. (1998) The air traffic flow management problem with enroute capacities. *Operations research*, 46(3), 406-422. DOI: 10.1287/opre.46.3.406.
4. Chan, T. M. (1996) Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, 16(4), 361-368. DOI: 10.1007/BF02712873. DOI: 10.1109/JPROC.2011.2161243.
5. Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimisation and Machine Learning*. USA: Addison-Wesley Longman Publishing Co., Inc.
6. He, T., Mantegh, I., Chen, L., Vidal, C., Xie, W. (2020) UAS Flight Path Planning for Dynamic, Multi-Vehicle Environment. In: *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, September 2020, IEEE, 211-219.
7. Hu, X. B., Wu, S. F., and Jiang, J. (2004) On-line free-flight path optimisation based on improved genetic algorithms. *Engineering Applications of Artificial Intelligence*, 17(8), 897-907. DOI: 10.1016/j.engappai.2004.08.015.
8. Kiss-Tóth, C., and Takacs, G. (2014) A dynamic programming approach for 4D flight route optimisation. In: *2014 IEEE International Conference on Big Data (Big Data)*, October 2014, IEEE, 24-28.
9. Kumar, V., Sherry, L., Kicing, R. (2009). Runway occupancy time extraction and analysis using surface track data. *George Mason University*.
10. Li, J., Wang, T., Hwang, I., & Hwang, I. (2009). A spectral clustering based algorithm for dynamic airspace configuration. In: *9th AIAA Aviation Technology, Integration, and Operations Conference (ATIO) and Aircraft Noise and Emissions Reduction Symposium (ANERS)*, September 2019, American Institute of Aeronautics and Astronautics (AIAA), 7056.
11. Liu, Y., Zhang, X., Zhang, Y., Guan, X. (2019) Collision free 4D path planning for multiple UAVs based on spatial refined voting mechanism and PSO approach. *Chinese Journal of Aeronautics*, 32(6), 1504-1519. DOI: 10.1016/j.cja.2019.03.026.
12. Ng, A., Jordan, M., Weiss, Y. (2001) On spectral clustering: Analysis and an algorithm. In: *Advances in neural information processing systems*, 2001, Cambridge: MIT Press, 849-856.
13. Odoni, A.R. (1987) The Flow Management Problem in Air Traffic Control. *Flow Control of Congested Networks*. NATO ASI Series, vol 38, 269-288. DOI: 10.1007/978-3-642-86726-2.
14. Sun, D., Clinet, A., Bayen, A. M. (2011) A dual decomposition method for sector capacity constrained traffic flow optimisation. *Transportation Research Part B: Methodological*, 45(6), 880-902. DOI: 10.1016/j.trb.2011.03.004.
15. Vaidhun, S., Guo, Z., Bian, J., Xiong, H., Das, S. K. (2020). Priority-based multi-flight path planning with uncertain sector capacities. In: *12th International Conference on Advanced Computational Intelligence (ICACI)*, August 2020, IEEE, 529-535.

16. Wen, H., Li, H., Wang, Z., Hou, X., He, K. (2019) Application of DDPG-based collision avoidance algorithm in air traffic control. In: *12th International Symposium on Computational Intelligence and Design (ISCID)*, December 2019, IEEE, 130-133.
17. Zhang, W., Kamgarpour, M., Sun, D., Tomlin, C. J. (2011) A hierarchical flight planning framework for air traffic management. *Proceedings of the IEEE*, 100(1), 179-194.
18. Zheng, C., Ding, M., Zhou, C., Li, L. (2004). Coevolving and cooperating path planner for multiple unmanned air vehicles. *Engineering Applications of Artificial Intelligence*, 17(8), 887-896. DOI: 10.1016/j.engappai.2004.08.016.
19. Zhou, H., and Hu, X. B. (2020) A Ripple Spreading Algorithm for Free-Flight Route Optimization in Dynamical Airspace. In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, December 2020, IEEE, 281-288.
20. <https://github.com/CascadingRadium/Air-Traffic-Distribution>.