LOW LEVEL DESIGN AND IMPLEMENTATION DOCUMENT

GPU – Accelerated Air Traffic Management and Flight Plan Generation

UE19CS390B – Capstone Project Phase – 2

Submitted by:

Rahul Rampure	**PES1UG19CS370**
Raghav S Tiruvallur	**PES1UG19CS362**
Vybhav K Acharya	**PES1UG19CS584**
Shashank Navad	**PES1UG19CS601**

Under the guidance of

Dr. Preethi P
Assistant Professor
PES University

August – December 2022

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

PES UNIVERSITY

LOW LEVEL DESIGN AND IMPLEMENTATION DOCUMENT

TABLE OF CONTENTS

1. Introduction

1.1. Overview

Air Traffic Management can be defined as the umbrella term for the technologies involved in assisting an aircraft to depart from an aerodrome, navigate the transit airspace, and land at a destination aerodrome smoothly. These technologies, termed as Air Traffic Services (ATS) and include Air Traffic Control (ATC), Airspace Management (ASM) and Air Traffic Flow and Capacity Management (ATFCM). This also entails an optimal scheduling of the flights themselves so that the airport traffic at the departure and arrival airports for the flights at the corresponding scheduled departure and arrival times is minimized. When a flight reaches its destination airport at a time when there is too much traffic there, then it's subjected to various holding patterns where they circle until it is their turn to land. However, aircraft flying in circles is an inefficient and costly way of delaying aircraft, so it is preferable to keep them on the ground at their place of departure, called a ground delay program. This way, the delay can be waited out on the ground with engines off, saving considerable amounts of fuel.

A Flight Plan is an air route that an aircraft desires to take before takeoff, in order to travel the transit airspace to reach its destination airport.

We aim to generate a flight plan for any flight while taking into consideration the existing traffic in the transit airspace and while striking a balance between the deviation suffered due to traffic and the amount of ground holding, essentially performing air traffic management. Hence, we plan on developing an algorithm that generates flight plans while taking air traffic management into consideration. The problem can be viewed as a congestion game with the goal being to approximate the Pure Strategy Nash Equilibrium of such a game.

1.2. Purpose

The objective is to generate flight plans which ensures optimal traffic flow such that traffic concentration in specific regions of the airspace is minimized or hotspots are eliminated as well as ensure that aerial times are minimized with an efficient and optimal ground holding policy. It comprises activities related to traffic organization and handling in a way that is safe, orderly, and expeditious. The necessity for a solution for path generation such that air traffic is considered stems from the fact that in current systems flight schedules are fixed by not a central authority but by the air carriers themselves based on a convenience factor and the plans are usually generated by taking only forecasted weather and aircraft aerodynamics into mind with no heed to air traffic management. Hence, we aim to merge the two ideas together and generate flight plans that lead to lesser overall airport and airspace traffic.

1.3. Scope

Our work focuses on air traffic management for decongestion of airspace to reduce the workload of the ATC. Since only one aircraft can land or depart from a runway at a given time, and because aircraft must be separated by a certain distance or time to avoid collisions, we specify that the number of aircraft inside a given airspace volume at a given point of time must be minimized. We propose a load-balanced multi path generation system such that the routes are distributed across the entire airspace in order to minimize traffic concentration while also considering the usage of a ground holding policy which delays the flight at the source airport in order to avoid traversing a longer aerial route while taking some hit in terms of cost depending on how much ground holding is done. Thus, we integrate flight plan generation with traffic management all the while accounting for an optimal ground holding policy.

2. Proposed Methodology / Approach

The approach chosen to achieve the goals specified above, we have chosen a customized genetic algorithm implemented in CUDA. The algorithm is executed on an input dataset which is all the flights that took place on a given day and the average traffic per unit ground surface area, flight duration and the ground holding time incurred is compared with the corresponding metric values for the actual flights. The algorithm takes in three input parameters – Population Size(P), Number of Mutations(M), Number of Generations(G).

2.1 Algorithm and Pseudo code

The airspace is divided into 1250 sectors by first performing spectral clustering on the cartesian coordinates of all the waypoints/fixes present in the US airspace. For each cluster obtained in the previous step, we use Chan's Algorithm to form convex hulls for each cluster. The hulls represent a sector and using empirical analysis we convert the entire network of hulls into a graph by considering a hull as a vertex and two adjacent hulls to be connected by an edge. We obtain the point at which the two hulls are said to be adjacent as the intersection of the line joining the centroids of the two adjacent hulls and the hull boundary. We find out such points for each edge and pass it on as an edge attribute in the graph.

The genetic algorithm consists of four operations namely, Selection, Crossover, Mutation, and Repair being called in a loop until the GA converges to a solution or reaches a maximum number of generations (G). The algorithm starts off with generating an initial population of P number of chromosomes or paths. The method used for generating initial population is Randomized DFS, where we spawn P threads on the GPU, and each thread generates a simple path from source to destination. The randomized DFS works as follows:

```
Cur ← Start
Path = List
Visited=Set()
While Cur is not End:
	Add Cur to Path
	Choose a random neighboring vertex to Cur which is not visited
	If(all neighboring vertices are visited)
		Refresh visited
		Clear Path and start over
	Mark the chosen vertex as Cur
Add Cur to Path
FindFitnessOfPath(Path)
```

The Fitness of a path is defined in terms of 4 factors Path Length (D), Path Delay(GD), Angle Cost(AD), Traffic Factor(T) with the fitness being defined as $\frac{1}{D*GD*AD*T}$. The distance is the actual path length, Path Delay is a value that ranges from [0,60) which entails that a plan/path starts at a delay of GD minutes and this corresponds to the ground holding policy of the GA. Angle Cost(AD) is the sum-total angle deviation the path suffers in terms of its sector boundary to sector boundary navigation. The Path is said to start at the centroid of the sector to which the start airport belongs, and traverses across the sectors using the point of contact present in the edge information between any two adjacent sectors and ends at the centroid of the sector to which the destination airport belongs. The Traffic Factor is the sum-total of all the number of other aircraft present in each corresponding sector of the path which the sector is being traversed by the aircraft for which the path is generated for. As the fitness is dependent on what ground delay the flight starts off with, each chromosome is assigned an array of fitness.

The Selection size(S) is fixed at (P/2), and we spawn S threads on the GPU, wherein each thread selects a chromosome from a pair of chromosomes. The selection done as follows, the average of all the fitness associated with both the chromosomes is found and the chromosome that has a higher average fitness is selected. The code is represented below:

```
Thread 1 assigned to Select from Paths indexed 2,6
Sum1=0,Sum2=0,Avg1=0,Avg2=0
For fitness in Fitness(2):
	Sum1+=fitness
For fitness in Fitness(6):
	Sum2+=fitness
Avg1=Sum1/60,Avg2=Sum2/60
If(Avg1>Avg2)
	Select 2
Else
	Select 6
```

The Crossover Size(C) is fixed at (S/2) where we spawn C threads where each thread crosses over two selected chromosomes. The crossover is done by finding common vertices between the two paths, and doing a single point crossover. These common vertices are found for the two paths by using an algorithm that ensures that the aircraft will be in the same sector at the same while taking into consideration the ground holding policy and the fact that the time spent in any sector is variable as it is determined by the distance travelled in that sector and the ground speed of the aircraft.

```
Thread 1 assigned to Crossover Paths indexed 2,6
CommonIndeces = FindCommonIndex(2,6)
Crossoverpoint=randomly choose from CommonIndeces.
Part of Path[2] right of Crossoverpoint is swapped with the Part of Path[6] right of Crossoverpoint.
If AverageFitness of Path[2] is lesser than before:
      Restore Path[2]
If AverageFitness of Path[6] is lesser than before:
      Restore Path[6]
Return Path[2],Path[6]
```

The Mutation is done by randomly selecting a set of (M) paths and for each path, selecting a random vertex, and starting a randomized DFS from that vertex to reach the end node, just as how it was done to get the initial population. M threads are spawned and each thread performs mutation on one chromosome.

```
Thread 1 is assigned to mutate Path indexed 2
MutationPoint = randomly select vertex in Path[2]
StartRandomPath(Path[2][MutationPath],End)
```

The Repair is done by spawning P threads, and each thread eliminates any cycles that might have crept up into the path due to the previous steps.

```
Thread 1 is assigned to repair Path indexed 2
RepairedPath=[]
RepairedPathIndex=0
For vertex in Path[2]
      if(vertex is in Path[2] at index X) //Cycle Exists
            RepairedPathIndex=X+1
      Else
            RepairedPath[RepairedPathIndex++]=vertex
```

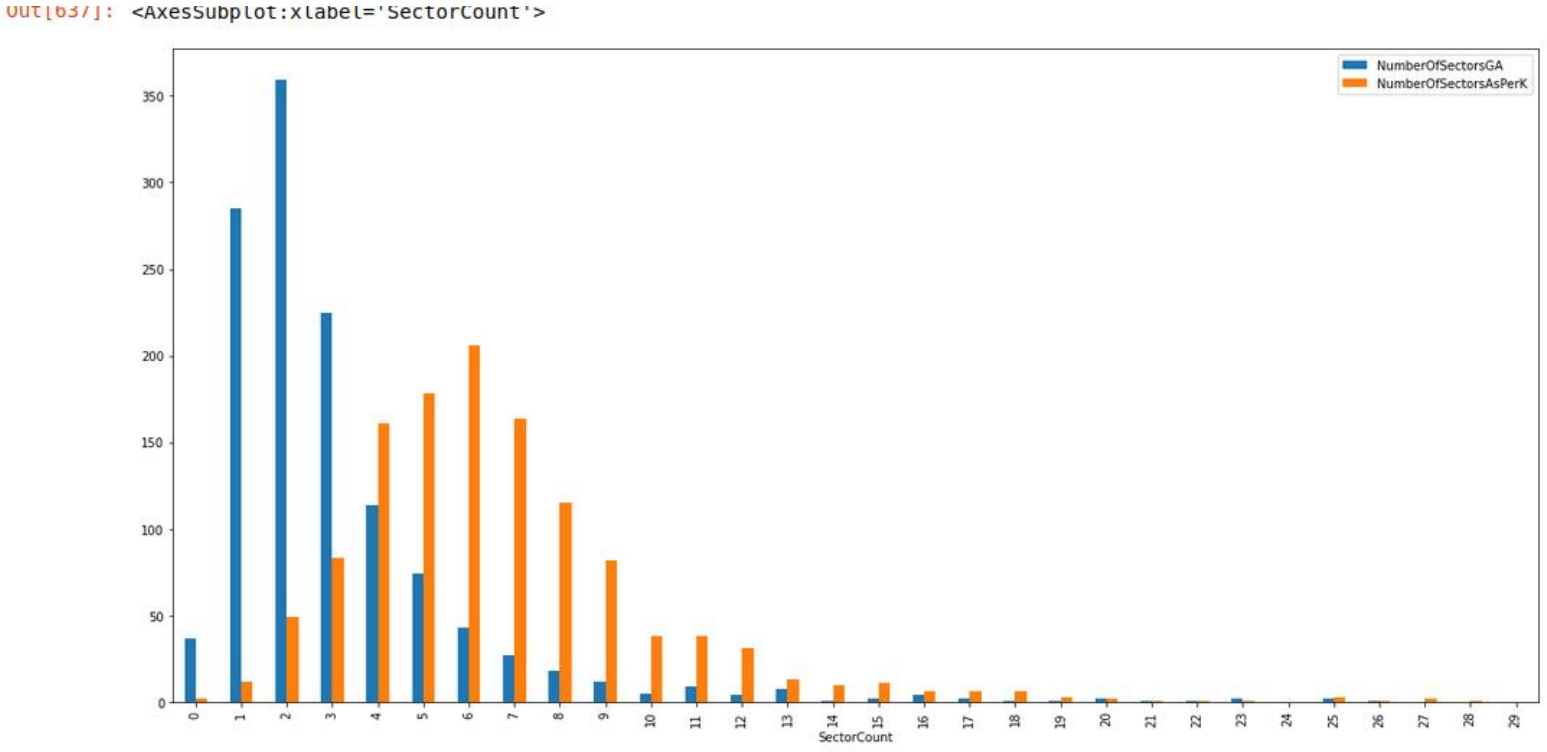
2.2 Implementation and Results

The three metrics chosen to evaluate the project is as follows:

The Aerial Time – Given the cruise speed (which we assume to be constant) what is the time the aircraft spends in the air as compared to the actual time spent by the aircraft in the air

The Ground Delay – The amount by which we delay the aircraft's departure as compared to the actual ground delay, which is the difference between the actual departure time and the scheduled departure time.

The Traffic Factor – The maximum number of aircrafts present in each sector at any point of time in the day is graphed against the real counterpart.



This graph aims to bring out the differences between the number of flights in each sector. The bars in orange are more skewed towards the right, indicating that majority of the sectors have high number of flights in their sector. The bars in Blue are skewed towards the left indicating that the flight plans generated chose such paths which minimize the number of flights in one sector.

The results we have obtained from running the standard Genetic Algorithm on python code for 8 Origin Destination pairs, the total time taken is 1512s=25 mins

```
NumberOfGenerations 46
Init Pop 8.623286962509155
Time for all gen 16.20020818710327 time 7
Prelim 2.297578811645508
Selection 2.4894471168518066
Crossover 2.8617727756500244
Mutation 0.14343857765197754
Repair 8.371365547180176
Elimination 0.036125898361206055
NumberOfGenerations 42
(6, 662)
```

```
GA TIME         1512.3031964302063
```

By re-basing our code in CUDA and C and making the complete architecture to run in parallel, we get a

```
sys     7m58.392s
ubuntu@ubuntu-ASUS-TUF-Gaming-A15-FA506IC-FA506IC:~/Desktop/Air-Traffic-Distribu
tion/CUDA GA$ time ./a.out

real    18m47.033s
user    7m35.277s
sys     11m12.137s
```

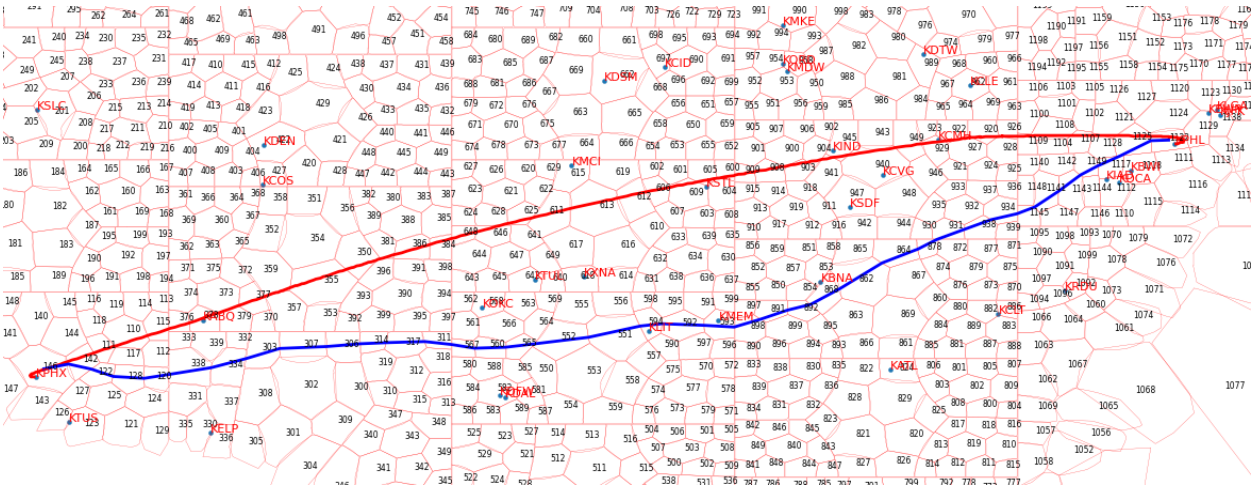
16x speedup


```
sys     7m58.392s
ubuntu@ubuntu-ASUS-TUF-Gaming-A15-FA506IC-FA506IC:~/Desktop/Air-Traffic-Distribu
tion/CUDA GA$ time ./a.out

real    18m47.033s
user    7m35.277s
sys     11m12.137s
```


It takes around 18 minutes to generate the path plans for 500 Origin Destination pairs.

GPU used to test is NVIDIA GeForce RTX 3050



The Real Path Taken by the flight is in Red and the path given by the GA is given in Blue. The Blue path is shorter than the red by almost 10 minutes. The Real data is obtained by using the Flight_Tracks and the Flight_History datasets for the month of August 2013.

2.3 Further Exploration Plans and Timelines

Convergence Criteria to be developed and Metrics to be found by running the code for a day in the month August 2013.

Appendix A: Definitions, Acronyms and Abbreviations

Air Traffic Services (ATS)

Air Traffic Control (ATC)

Air Traffic Flow and Capacity Management (ATFCM)

Population Size(P)

Number of Mutations(M)

Number of Generations(G)

Path Length (D)

Path Delay (GD)

Angle Cost (AD)

Traffic Factor (T)

Selection size (S)

Crossover Size (C)

Appendix B: References

A Genetic Algorithm for Shortest Path Routing Problem and the Sizing of Populations

Chang Wook Ahn; R.S. Ramakrishna

IEEE Transactions on Evolutionary Computation : Vol 6 Issue 6