A Dynamic Programming Approach for 4D Flight Route Optimization

Christian Kiss-Tóth, Gábor Takács
Department of Mathematics and Computer Science
Széchenyi István University
Győr, Hungary
Email: ktchris@sze.hu, gtakacs@sze.hu

***Abstract*—This paper describes our solution for the GE Flight Quest 2 (FQ2) challenge, organized by Kaggle. FQ2 aimed at optimizing flight routes so that the overall cost depending on fuel consumption and delay is as low as possible. The contestants could use several data tables as inputs, including aircraft positions and destinations, weather information and other aviation related data. Their task was to produce a flight plan for each flight, given as a list of (latitude, longitude, altitude, airspeed) quadruplets. The cost of the flight plans was evaluated with an open source simulator. Our proposed method produces an initial solution with the Dijkstra's algorithm to avoid restricted zones, and then refines it using dynamic programming and local search techniques. We can extensively utilize wind forecasts and significantly divert the planes from the the great circle route if necessary. Moreover, our method tries to set the ascending and descending profiles of the flights to further decrease the cost. Our algorithm achieved second place on the public, and fifth place on the private leaderboard of the contest.**

***Keywords*—GE Flight Quest, route optimization, dynamic programming**

I. Introduction

Economic growth makes air transportation more important every day. The increasing number of flights leads to high complexity scheduling and planning problems. Airlines are constantly seeking ways to make the flights more efficient. From gate conflicts to operational challenges to air traffic management, the dynamics of a flight can change quickly and lead to costly delays. In the contests GE Flight Quest 1 and 2, organized by Kaggle, the aim was to solve two kind of problems related with aviation.

In GE Flight Quest 1 (FQ1) [1], the goal was to predict the runway and the gate arrival times of U.S. domestic flights. Predicting the arrival of the flights is important to avoid congestion at the airports, which would lead to costly parkings.

GE Flight Quest 2 (FQ2) [2] was a flight route optimization challenge. In a commercial airline, flight route planning is a central element of the overall planning process. Small adjustments to the plans can add up to substantial savings for the airline. This paper deals with FQ2, and describes our solution that finished second on the public and fifth in the private leaderboard.

In the FQ2 competition, the contestants obtained input for their algorithm in several data tables, containing current aircraft positions and destinations, weather forecast information and other aviation related data. The input data was given up to a "cut off" time on each test day. The task of the contestants was to produce a flight plan for each flight, so that average the cost of the plans is as low as possible.

The cost of the plans was evaluated with an open source simulator and it depended on fuel consumption, delay and some other factors. The contestants could only access forecast weather data for creating their plans, but the final evaluation was based on live weather data. From a mathematical point of view, the problem to solve in the FQ2 contest was the constrained minimization of a noisy and known cost function.

A flight plan in FQ2 was a list of (latitude, longitude, altitude, airspeed) quadruples. This considers three space dimensions plus a a time dimension, therefore we can call it a 4D flight plan. An example flight plan for one plane is shown on Table I.

TABLE I: Flight plan for one plane containing 5 instructions. The altitude is given in feet, the airspeed is given in knots.

ID	Ordinal	Latitude	Longitude	Altitude	AirSpeed
32441974	1	37.07	−109.73	40000	600
32441974	2	36.86	−110.35	40000	600
32441974	3	35.23	−115.03	40000	600
32441974	4	34.43	−117.14	2000	600
32441974	5	34.40	−117.23	2000	600

Flight plans not only have to take the laws of physics into account but also route restrictions from the air traffic control and aviation authorities. This means that there are geographical zones in which the planes must not fly, or there are specified altitudes for different types of flights where they are allowed to cruise.

A good flight plan should (1) find a short 2D path between the origin and a destination with taking winds into account, (2) find a proper cruising altitude and establish an efficient altitude profile for ascent and descent and (3) set the airspeed to keep a balance between traveling time and fuel consumption.

The rest of the paper is organized as follows: First, we briefly overview related work. Second, we dig into the details of the input data and the simulator. Third, we present our solution and show numerical results. Finally, we summarize and conclude.

II. Related work

Although the general area of route planning is old, not many algorithms have been published yet for the 4D flight

route planning problem. This is mainly because the realistic evaluation of a 4D flight plan requires a highly complex simulator environment. No such environment was publicly available before the FQ2 contest to the best of our knowledge.

One work that copes with the 4D version of flight route planning is [3] by Boeing. They do not disclose too many details of their method, but outline an approach that applies dynamic airborne replanning by evaluating multiple plans, and takes forecast winds and temperatures into account. Moreover, they also optimize the amount of departure fuel besides the 4D flight plan.

Bousson and Machado [4] investigate 4D flight trajectories from a control theory point of view. They propose pseudospectral methods for following a prescribed 4D flight trajectory so that the arrival delay at the waypoints is small. Although, this is an interesting topic, in the case of FQ2 we can assume that flights can follow trajectories reasonably well.

The MSc thesis of Merle [5] gives a good overview on simulating the fuel consumption of aircrafts. The author does not give an algorithm for route planning, but gives a complex mathematical model for aircraft fuel consumption and analyzes the cost efficiency of a few sample flights in detail.

The work of Atkins [6] outlines a framework for flight plan management. It focuses on the architecture of the software and practical questions related to real-life application (e.g. how is the aircraft connected to the data sources or how can the pilot interact with the system). Although the system contains an flight planning module, no details are disclosed about it.

III. Input Data

This section gives more details about the input data available for the route planning algorithms.

A. Flight Data

- Status information about flights for that the route to the destination had to be optimized.
- Data was provided for 14 days for approximately 1000 flights on each day.
- For each flight, the departure and arrival airport, the current position and altitude, the departure and scheduled arrival time, fuel, delay, turbulence costs and some other, less important, information was given.

B. Airports

- Latitude, longitude and altitude of 63 airports where the planes had to land.

The airports and the positions of the planes at the cut off time of an example day is shown in Fig. 1.

C. Restricted Zones

- Planes were not allowed to fly into the restricted zones of the airspace.
- Each zone was a convex polyhedron, given with the coordinates of the vertices and the lower and upper altitude bounds.
- Restricted zones were the same on every day.
- The number of zones was 19.

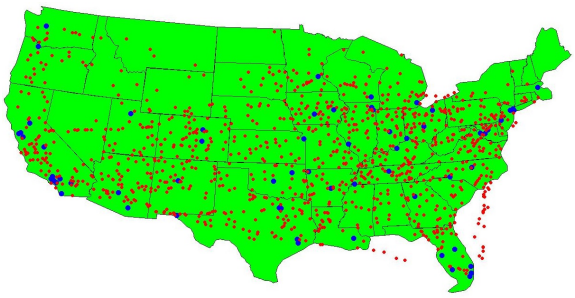

Fig. 1: Airports (blue) and plane positions at the cut off time of an example day (red).

D. Turbulent Zones

- Planes got some cost penalty, if they flew the turbulent zones of the airspace.
- Each zone was a convex polyhedron, given with the coordinates of the vertices and the lower and upper altitude bounds.
- Turbulent zones were different on different days.
- The number of zones varied between 5 and 9 zones per day.

E. Weather Data

- Wind data was provided as 2D vectors given hourly at eight altitude levels an a 451×337 geographical grid.
- There was two kinds of wind data: live wind before the cut off time, and forecast wind from the NWS SOO Science and Training Resource Center for the entire flying period.
- Some other weather data was also given for the airports: ground conditions for the arrival, such as temperature, wind and visibility, given in hourly resolution up to the cut off time.

The polygons of the restricted and turbulent zones for an example day, and the wind vectors on the grid of the highest altitude level is shown in Fig. 2.

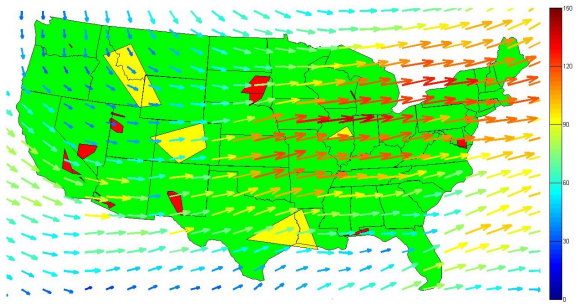

Fig. 2: Restricted zones (red) turbulent zones (yellow) and wind vectors on the highest altitude level.

F. The Simulator

As mentioned earlier, the objective function of FQ2 was calculated by a simulator. The simulator was written in F#,

and it contained ~ 2300 lines. The source code was disclosed for the contestants.

The simulator consists of several submodules, including fuel consumption models for ascending, cruising and descending, a landing, weight, atmosphere and aircraft model and an airspeed limiter. Some of the modules implement physical laws, others are statistical models with parameters fit to measurement data.

The simulator computes a total cost value for all flight plans loaded into it, and finally outputs one number, the average total cost. The structure of the cost function for one flight can be written in the following form:

$$C_{\text{total}} = C_{\text{fuel}} + C_{\text{delay}} + C_{\text{oscillation}} + C_{\text{turbulence}}$$

The fuel cost is proportional to the burned fuel, the delay cost is the sum of a linear and a nonlinear term, the turbulence cost is a linear function of the elapsed time in turbulent zones and the oscillation costs penalize if there are to many altitude changes during the flight.

Simulating a flight plan means a discrete simulation with a time step of 6 minutes, following the instructions. A flight can take three kind of steps: *ascending, descending* and *cruising*.

During the descending steps, the fuel consumption is a function of the altitude of the aircraft:

$$\text{fuel}_{\text{descent}} = f_1(\text{altitude}).$$

During the ascending steps, the fuel consumption is a function of the altitude and weight:

$$\text{fuel}_{\text{ascent}} = f_2(\text{altitude}, \text{weight}).$$

During the cruise steps, the fuel consumption also depends from the airspeed:

$$\text{fuel}_{\text{cruise}} = f_3(\text{altitude}, \text{weight}, \text{airspeed}).$$

One more relevant characteristic of the simulator was that the ground speed of the aircraft was calculated as the function of airspeed and wind:

$$\text{speed}_{\text{ground}} = g(\text{airspeed}, \text{wind}).$$

IV. Our Proposed Solution

We split the problem into three main parts. First, we create an initial flight plan for the aircraft. Second, we perform a 2D optimization process, to set the latitude and longitude coordinates of the waypoints. Third, we set the altitudes and the airspeed of the flight along the route obtained from the previous step. The 2D and the 1D profile of a flight plan is visualized in Fig. 3.

A. Initial Routes

For hitting any of the restricted zones of the airspace (e.g. Edwards Air Force Base), the simulator gives a penalty of $100,000$\$ per plane, and in some cases the shortest (great circle) path hits some of the zones. The first step of our method

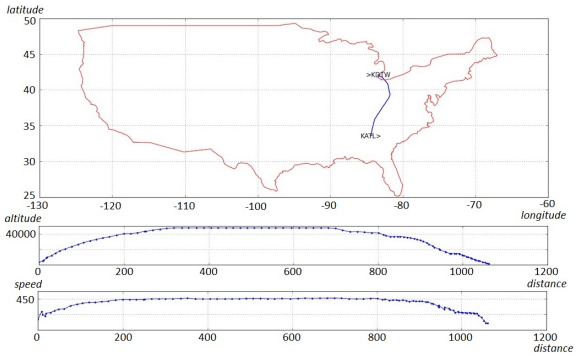

Fig. 3: The 2D shape, the altitude-distance and the velocity-distance graph of an example flight.

is to create initial solutions avoiding these incidents. Since the restricted zones are convex polyhedrons, we can translate this problem into a simple geometry question: how can we connect two points on a plane (current position, destination) if we have to avoid a set of convex polygons? Of course, to create good initial routes, we would like to find the shortest path. This problem can be solved with Dijkstra's well known algorithm.

The **Dijkstra's algorithm** [7] computes the shortest path from one point to all others in an edge-weighted graph with non-negative weights. In our case, the points of the graph are the current position, the destination airport and the vertices of the restricted zones. The weights are the distances between the points. With the 2D paths given by the algorithm we can guarantee that none of the planes will hit a restricted zone. Creating an initial altitude and airspeed profile also, we are already able to make a reasonable initial plan for all flights. In Fig. 4, the Dijkstra's path for an example plane is shown for that the great circle path would hit two of the restricted zones[1].

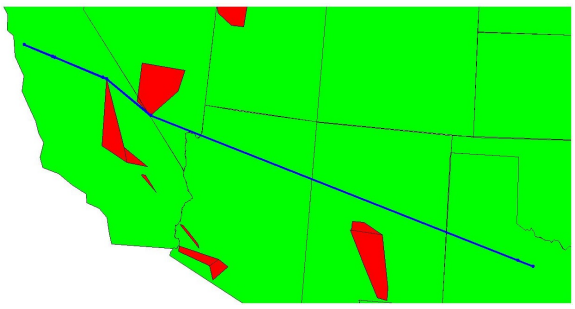

Fig. 4: An initial 2D path with two additional waypoints created with Dijkstra's algorithm.

B. 2D Optimization

In the cost model, the burned fuel is a function of the airspeed, but the ground speed is the sum of the velocity relative to the air and the wind vector. This means that for some flights a wind-optimal path may be far from the great circle. Taking advantages on the wind can significantly reduce the fuel

[1]Maps are drawn using Lambert coordinates, which means that great circles are straight lines in these figures.

cost, and since the fuel consumption is directly proportional to the elapsed time, it can also decrease the delay costs. Turbulence cost can be also reduced in these cases. In Fig. 5, a 2D profile of a flight is shown, where the wind-optimal path is far from the great circle and significantly reduces the fuel, delay and turbulence costs.

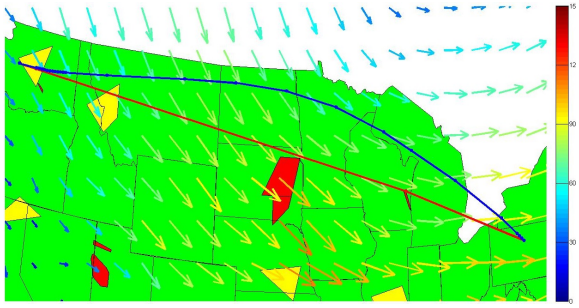

Fig. 5: A wind optimal 2D path far from the great circle can significantly reduce the fuel, delay and turbulence costs.

For the flight above, the blue route reduces the total cost with nearly 15 percent. The core part of our solution was to explore the airspace to discover such routes with a **dynamic programming** based approach.

Our algorithm creates a grid in the airspace. It divides the initial path into N equal parts with waypoints, and determined n points perpendicular to the initial path after d miles in both direction. We call these $2n+1$ points "levels", and we have N levels if we call the current position level 0, and the destination airport level N. We denote the grid points with p_{ij}, where $1 \leq i \leq N-1$ and $-n \leq j \leq n$, and p_{00} is the current, and p_{N0} is the destination point. N, n and d are parameters of the algorithm. In Fig. 6, an example grid is shown with parameters $N = 6$, $n = 2$, $d = 50$.

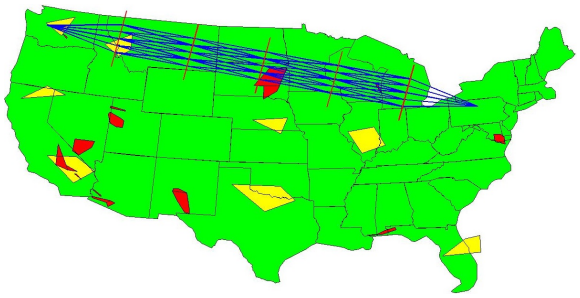

Fig. 6: Grid points for dynamic programming with parameters $N = 6$, $n = 2$, $d = 50$.

The original objective function cannot be evaluated for partial routes, as the delay cost is not defined. Therefore we created one by omiting the delay term from the formula of the total cost.

We recursively determined the cheapest route from the current position to every point of the grid. Each point had three labels:

- Cost_{ij}: The lowest possible cost to reach p_{ij} from p_{00}.
- Prev_{ij}: The index of the previous point at level $i-1$ in the best route.
- Prop_{ij} The status of the aircraft after reaching p_{ij} in the best route.

During the algorithm for all grid points p_{ij} we simulate the routes from all the points p_k in the previous level, with starting properties Prop_k and register the best one (to make the notation shorter, we use the single index k instead of $(i-1,k)$ for the points in the previous level). Apart from level 1 this is the minimum of $2n+1$ possible routes. At the end of the procedure we obtain a new route that is not worse then the initial one since the initial path was also present among the ones we considered. The pseudo-code of our dynamic programming based route optimizer is shown in Fig. 7.

▷ Procedure to construct wind-optimal routes with dynamic programming
function DYNAMIC 2D OPTIMIZATION(N, n, d)
  $\text{Cost}_{00} := 0$
  $\text{Prop}_{00} :=$ (initial position, initial weight, initial time)
  for $i = 1$, N **do**
    for $j = -n$, n **do**:
▷ Simulate path to p_{ij} from all the points p_k of level $i-1$ and store the arrival properties
      $\text{Cost}_{ij,k} :=$ total cost across point p_k
      $\text{Prop}_{ij,k} =$ (position, weight, time)
▷ Find the best path to p_{ij}
      $\text{Cost}_{ij} := \min_k C_{ij,k}$
      $\text{Prev}_{ij} := k$
      $\text{Prop}_{ij} = \text{Prop}_{ij,k}$
    end for
  end for
return *Optimized path*
end function

Fig. 7: Pseudo-code for the dynamic programming phase.

After the the dynamic programming phase, we further enhanced the 2D route with local search procedures. This means that we perturbed the waypoints of the routes in several directions and kept the perturbation if it reduced the cost. We call this part of our algorithm **2D refinement**.

C. 1D Optimization

After we created 2D paths, the next step is to optimize the altitudes (ascending, cruising and descending part of the flight) and the speed profile. Since the oscillations (altitude changes) are also highly penalized - 100$ per extra oscillation – we divided every flight into three parts: to an ascending, a cruising and a descending part. Analyzing the cost models for the different flight profiles we found that the most critical part of the planning is to determine the starting point of the descending phase. Since the 2D profile is fixed, this point can be given with it's distance from the destination airport.

Our initial flight profile was the following: if the flights were far enough from the destination they ascended to the maximum altitude level, cruised, and started descending. As we described it in the Simulator subsection, the airspeed instruction affects only in the cruising phase, so there was no reason the decrease the airspeed from the maximum in the ascending and descending phases. In some cases it is worth to decrease the airspeed during the cruising part: if the plane

is on time, which means the delay cost would be still zero even with a lower cruise speed, it is possible to reduce the fuel consumption by decreasing the speed.

This means that the parametrization of the 1D profile can be done with two variables: the first one is the *descending distance* and the second one is the *cruise speed*. For this **1D optimization** part we implemented an exhaustive search method to optimize these two constants. For some of the flights (an example for the optimized 1D profiles plotted on Fig. 8) this improvement spared another 10 percent of the total cost:

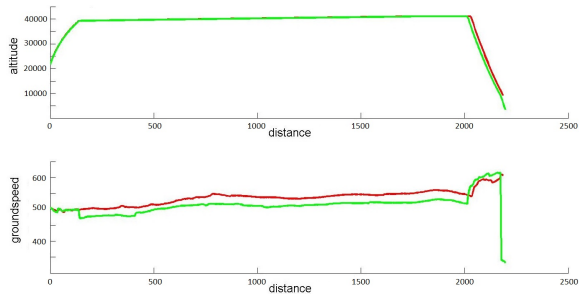

Fig. 8: Optimized altitude-distance and groundspeed-distance profiles for a plane arriving on time.

During the 1D optimization process, we found an extra possibility to reduce the cost. Due the articacts of the simulator, it was worthwhile to insert extra waypoints into the ascending and descending phases of the plans just to increase the density of waypoints in these phases. Of course this **1D refinement** procedure is effective only in the FQ2 simulator, it would have no impact in real life.

D. Implementation Details

We used Bash scripts for automation, and Python for data processing. We rewrote the FQ2 simulator and implemented the dynamic programming based approach in MATLAB. We divided the flights on each day into 4 parts, and ran the optimization process on $14 \cdot 4 = 56$ cores. The hardware we used during the competition was a 64 core Linux server with 1 terabyte memory.

V. Numerical Results

In this section, we compare the efficiency of different parts of our solution. Table II shows objective function values for the solutions created after each step.

TABLE II: Reduction of the average cost after different steps of our algorithm.

Solution type	Average cost	Improvement
Dijkstra's algorithm	11736.4$	—
Dynamic programming	11707.4$	29$
2D refinement	11702.3$	5.1$
1D optimization	11687.2$	15.1$
1D refinement	11680.5$	6.7$

The time requirement of the steps is depicted in Fig. 9.

The entire space requirement of our optimizer was 11.5 GB. Comparing the cost reduction table with the time requirement chart we can make some observations:

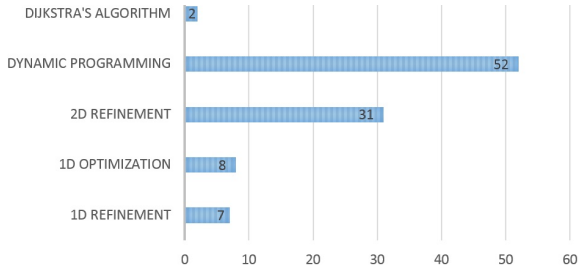

Fig. 9: Time requirements of the different phases of the algorithm, measured in hours.

- Creating a crash free initial solution is fast.
- 2D optimization is the most "intelligent" part of the algorithm. It is time consuming, but reduces the cost to a great extent.

VI. Conclusion

This paper presented our solution that reached fifth place in the final phase of the GE Flight Quest 2 competition. The main elements of our route optimization method are the Dijkstra's algorithm, dynamic programming, local and exhaustive search procedures. Our method is able to produce reasonable initial plans in a short time. To improve the initial solution the most effective and time consuming part was the dynamic programming part. Most of our methods can be useful for real life flight route optimization, since the simulator used in FQ2 was quite realistic in many aspects.

Acknowledgment

This paper was supported by TÁMOP-4.2.2.C-11/1/KONV-2012-0012: "Smarter Transport" – IT for co-operative transport system – The Project is supported by the Hungarian Government and co-financed by the European Social Fund.

References

[1] "GE Flight Quest 1," https://www.gequest.com/c/flight.

[2] "GE Flight Quest 2," https://www.gequest.com/c/flight2-final.

[3] S. Altus, "Effective flight plans can help airlines economize," 2009, http://www.boeing.com/commercial/aeromagazine/articles/qtr_03_09/article_08_1.html.

[4] K. Bousson and P. Machado, "4d flight trajectory optimization based on pseudospectral methods," *Engineering and Technology*, vol. 4, pp. 471–477, 2010.

[5] D. Merle, "Flight path optimization for an airplane," Master's thesis, Norwegian University of Science and Technology, 2011.

[6] E. M. Atkins, "Flight Plan Management with George Jetson as Pilot," in *AAAI Spring Symposium*, 2004.

[7] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.