Deloitte.

Quantum.Link by Deloitte

Quantum Climate Challenge 2022

Quantimize

Kevin Shen
Jakob Pforr
Franziska Wilfinger
Jezer Jojo
Catharina Broocks

May 17, 2022

Quantum Climate Challenge 2022 *Quantimize*

Reducing the negative impact of air travel on the climate by optimizing flight trajectories

SUMMARY

In the Deloitte Climate Challenge, the task was to explore how the contribution of air travel to the anthropogenic climate change can be reduced by optimizing flight trajectories using quantum, quantum-classical hybrid, or quantum inspired solutions. Therefore, several flight paths with different schedules and a map showing the climate impact depending on the flight level and airplane position are given. Additionally basic safety regulations are taken into account. With respect to all these constraints the flight trajectories are optimized so that the overall negative climate effect is minimal.

In our work we implemented the nowadays fuel-efficient straight line solution and a classical genetic algorithm. Moreover, we discussed three types of quantum solutions: a Quantum Approximate Optimization Algorithm (QAOA) for Quadratic Unconstrained Binary Optimization (QUBO) problem, the quantum genetic algorithm and a quantum neural network.

We found out, that at the moment the classical genetic algorithm is the best choice for a big data set. However, we are confident, that this will change with larger and better Noisy Intermediate-Scale Quantum (NISQ) hardware.

The most promising quantum algorithm we found is to discretize the space, transform the cost function into QUBO form and then solve it by quantum annealing or the QAOA algorithm on gate-based quantum computers such as IBM's superconducting quantum devices. A four-qubit device is enough to build a scaled-down problem for demonstration purpose. However, more qubits are needed to run it on the big data set to avoid oversimplification.

The quantum genetic algorithm performs solid, but does not give an obvious advantage over the classical genetic algorithm.

The quantum neural network (QNN), we implemented, is not trainable with the currently available number of qubits. However, we do believe that this is a promising concept, which can be useful in the future, when larger devices are standard.

Contents

1 The team behind Quantimize

Kevin Shen
Quantum Science & Technology M.Sc.
Technical University of Munich
https://www.linkedin.com/in/kevinshen-tum/

Jakob Pforr
Quantum Science & Technology M.Sc.
Technical University of Munich
https://www.linkedin.com/in/jakob-pforr-a74a78210/

Franziska Wilfinger
Quantum Science & Technology M.Sc.
Technical University of Munich
https://www.linkedin.com/in/franziska-wilfinger/

JezerJojo
BSMS
Indian Institute of Science Education and Research Pune
https://www.linkedin.com/in/jezer-jojo/

Catharina Broocks
Quantum Science & Technology M.Sc.
Technical University of Munich

Hi, we are Quantimize!
We all are young and ambitious students interested in quantum computing. We already knew each other from the Global Qiskit Hackathon and were glad when an opportunity showed up to work together on an interesting quantum topics again.

2 Introduction

Climate change is one of the greatest challenges of our time and it needs our full attention to be solved as fast as possible to prevent catastrophic conditions [1]. 2.4% of global CO2 emissions come from aviation (2018). Flights also induce non-CO2 effects, which can also lead to global warming. The non-CO2 effects depend on the time of day, the position and the weather and could therefore be reduced with good flight route planning [2]. This is exactly what we have done in this work. We tried to find the best way to minimize the climate impact of aviation by taking CO2 and non-CO2 effects into account. For that we wanted to find the most powerful algorithm, which can solve this optimization problem.

Our work is structured into four major parts. As a first step we cleaned the data into forms that are easily accessible for the coming optimization algorithms (see section 4). We also developed tools for evaluating trajectory outputs and visualising data.

Afterwards we began by searching for the best classical solution (see section 6). We saw it has solid performance. However, we thought there is room for improvement and quantum algorithms, with many possibilities and resources, like entanglement and superposition, could potentially outperform it.

Since climate change is an urgent issue, as already stated above, we focused on NISQ era quantum algorithms or quantum inspired algorithms. We especially look for quantum algorithms that could solve the problem with an order of 1,000 high quality qubits, which is likely to be reached in the next decade [3]. We came up with several ideas for quantum algorithms, which different strengths and weaknesses. More information and a discussion about our quantum solutions can be found in section 7.

In the last step we benchmarked our algorithms against each other and discussed the results. This can be found in section 8.

To store and provide our code, we created a GitHub repository called „Quantum Challenge “:
`https://github.com/fjelljenta/Quantum-Challenge`

In the repository you can find a file called „requirement.txt“, which provides all the packages necessary to run our code. The package Quantimze contains all the functions we have written. The „Quantimize.ipynb“ is our main program and summarizes everything. Here you can find our classical and all our quantum solutions plus a benchmarking. The notebook „Tutorial.ipynb“

explains how to call and use our functions if you would like to play around with our code on your own.

3 Problem definition and key concepts

It is important to first define the problem precisely. We are facing an optimization problem with constraints, like safety regulation, boundaries of space, constraints on the turning, descending, and climbing rate of the plane, etc. Given these constraints, there are still infinitely many allowed trajectories for a flight. Some assumptions on the shape of the trajectory must be made to allow parametrization, and hence to enable the usage of optimization algorithms. Continuous variables are best suited to describe this problem since in real world planes usually follow smooth trajectories. Smooth analytical functions like B-spline curves, parameterized by continuous variables, is a very good option. The cost function under this model is expected to have a complicated landscape in parameter space, because it is a function of the atmospheric data and the trajectory, neither of them has a simple mathematical form. Nevertheless, simplified models with discrete variables can also be considered.

We require our optimization algorithms to take inputs, up to some hyperparameters, and outputs of the same form.

$$\text{Algo: Flight No.} \mapsto [(\text{long}_1, \text{lat}_1, \text{FL}_1, \text{time}_1), \ldots, (\text{long}_N, \text{lat}_N, \text{FL}_N, \text{time}_N)]$$

Each algorithm takes in the flight number, with which the corresponding starting time and position and the destination are easily accessed. The algorithm should then find the optimal flight path in between. Optimal is, where the climate cost is minimized. The output is a trajectory list containing information about the flight coordinates at different times. This trajectory can then be mapped onto a 3D plot for visualisation. From the trajectory the climate cost, as well as the fuel consumption and the flight time can be computed. Especially the later ones seem to be interesting for the flight companies, while the first one gives information about fighting climate change.

Before we continue with solving this problem we would like to introduce some key concepts, which are later important for understanding our solutions:

The trajectory object is a list containing the flight coordinates (longitudinal, latitudinal and flight level) for defined time stamps. The time stamps can

be modified by us. With that we can choose time stamps being a compromise of the fines of the trajectory and the computational cost. The trajectories are not only the output of our optimization, the also are the input for all kind of following computations, like the calculation of the climate cost, the fuel consumption, the safety check and many more.

$$[(\text{long}_1, \text{lat}_1, \text{FL}_1, \text{time}_1), \ldots, (\text{long}_N, \text{lat}_N, \text{FL}_N, \text{time}_N)]$$

The climate cost/climate impact is the overall environmental impact taking all CO2 and non-CO2 emissions into account. It is given by the total rise in temperature in Kelvin for each flight i and voxel j

$$\Delta T = \sum_{i,j} (\Delta C_{\text{non CO2}}(i,j) + \Delta C_{\text{CO2}}) \cdot F(i,j)$$

F(i,j) is the fuel of flight i consumed in voxel j. The ΔC's are the temperature rise for non-CO2 and CO2 effects. [4]

The straight line solution is the straight line connecting the start and the end coordinates for a given flight. No changes in the altitude are considered. This solution is in fact the simplified version of the "great circle trajectory" in real world after neglecting the curvature of earth surface. It is seen as the standard fuel-efficient solution used by airlines. It is also used as a benchmark for our solutions to see how the climate impact can be reduced. The implementation can be found in GitHub under quantimize/classic/classic_toolbox.py.

The B-spline curve is used for the computation of a curved solution. B-spline curves are defined as a piecewise polynomial function with minimal support [5]. The mathematical description is:

$$r(t) = \sum_{i=0}^{n} p_i N_{i,k}(t),$$

where p_i are the control points and $N_{i,k}$ the B-spline basis functions. The idea behind using these is to determine a unique polynomial representation of points in the 3D space.

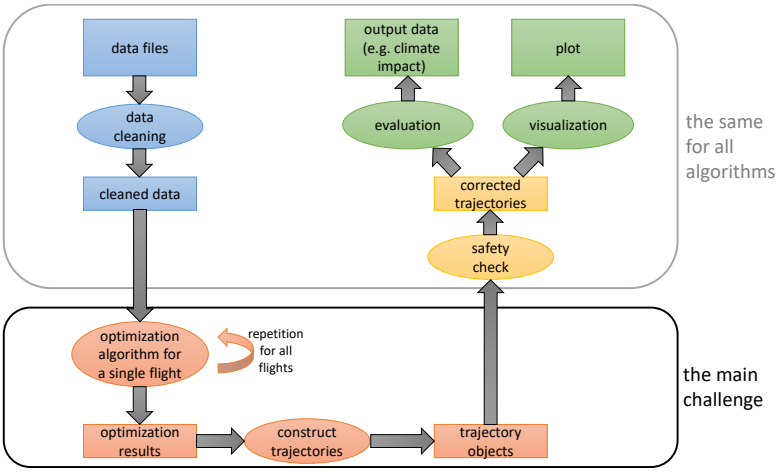

Figure 1: Flowchart of our project showing all the relevant processes and outcomes. The top part (grey rectangle) works the same for both classical and quantum solution. The bottom part (black rectangle) is the main challenge, especially to provide a quantum solution.

4 Data preparation and visualization

As one can see in the flowchart in Figure 1, the project can be divided into two parts. One part is the preparation of the data and also the way to deal and proceed with the output trajectories. This part is the same for the classical and quantum solutions. The second part, which we called „main challenge“ in the flowchart, is the actual optimization algorithm. To fill this part with life was the task of the challenge. However, the first part is also very important and will be presented in this and the next section.

We started by extracting and cleaning the data to bring them in an easily accessible form. To access the data „data_access.py“ is used. The two most important function of this document are „get_flight_info“ and „get_merged_atmo_data“. The former returns a dictionary containing the flight information for a given flight number. The later returns the merged climate impact for a given set of location, flight level and time. Another basic code file, which was very useful for several parts of our solution, is the „converter.py“, which converts between different units. Moreover, it also includes functions, which calculate the distance if only start and end coordinate are given.

In order to perfectly understand the data and get an intuition about the problem, we visualised the atmospheric data for different flight levels and

times. One example can be seen in figure 2.

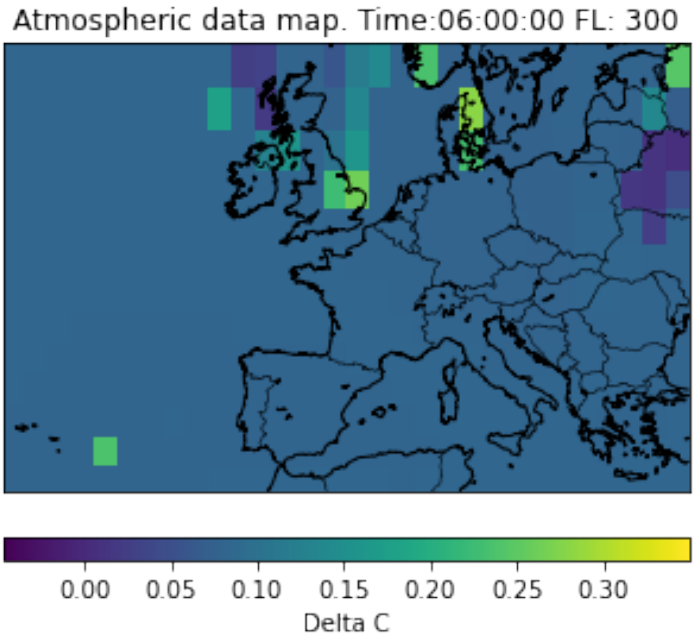

Figure 2: Atmospheric data plotted in different colours corresponding to different climate impact values. In the title, you can also see, that the data correspond to flight level 300 at 6 o'clock in the morning. The data stay the same for a certain period of time (6am-9am, 9am-3pm, 3pm-6pm). Moreover, we do not have atmospheric data for all flight levels, so they are mapped to their nearest available neighbour, e.g we have no atmospheric data for FL 295, so we choose the closest, which is FL 300.

In addition to that we also provided a 3D visualization, where the flight curves including height changes can be seen perfectly. The colour of the trajectory corresponds to the atmospheric data. One example can be seen in figure 3. It shows the straight line solution and the curved solution generated by our genetic algorithm of flight 17.

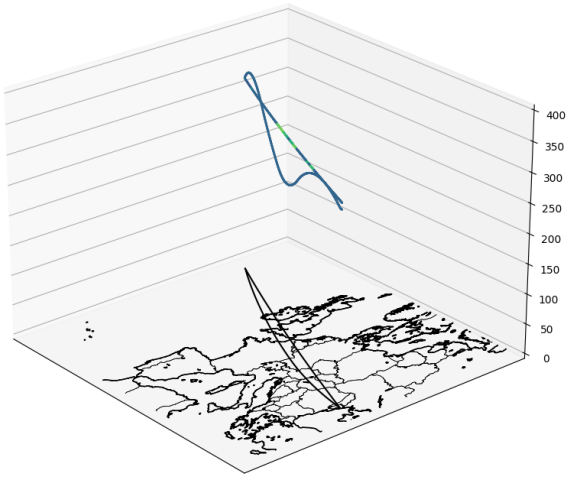

Figure 3: Flight trajectories of flight number 17. Once calculated with the straight line solution and once with the genetic algorithm. The colour of the line corresponds to the atmospheric data.

5 Air security

Our air security check is applied via the function „safe_algorithm_2 “ as an outer loop over the trajectory creation. It takes a list of flights and an optimization algorithm as input, computes the optimized trajectories with this algorithm and checks for violations regarding the safety distances and curvature radius (detailed description below). If any violations are detected, the associated trajectories will be corrected by re-calculating them. This procedure repeats for several times, after that a list of optimised, non-crashing trajectories is returned. If some violations can't be solved after that time, the trajectories are corrected with respect to safety regulations as good as possible and the user is asked to change the starting times or levels of the flights which cause crashing or curvature radius problems.

Our function „check_safety “ is responsible for detecting violations. This function checks if:

- a plane distance of at least 5NM is hold
- an out of plane distance of at least 10 flight level is hold

The safety regulations are met, when the conditions above are fulfilled for every flight. To do this, we first map our trajectories on a time gird, so we can check for every point in time for collisions. The check function returns an empty list if the regulations are met for all time steps and a list of conflicts

if the regulations are not met. In order to work properly it is important to choose appropriate time steps. We choose 15 seconds, since the maximal true air speed is 459kt/s and the minimal distance is 5NM in horizontal and 1000feet in vertical direction. With the maximum air speed, the plane would need 39.2 seconds to travel 5NM, which is way more than our chosen 15s, so it is nearly sure that we will find a point failing the safety check. The maximal rate of climb/ descent is 3830 ft/min. So, in 15s the maximal change in height would be 957.5 feet, which would be definitely detected in a next step, if another plane would stay on the same level. Therefore, a detection of a violation with a time step of 15s is highly likely. However, it stays the possibility that in really rare cases with hard curves the security check will fail. To make it more secure we would need to decrease the time steps, which would make the whole computation more costly and more time consuming. Nevertheless, we think for our purpose 15s are enough, since our flight trajectories change quiet smooth.
In addition to the check safety, we also continuously check the radius of the trajectory via the function „radius_control “. This function checks

- that the airplane does not make tighter turns than 25°

6 Our classical solution

In order to find the best classical solution, we read the papers mentioned in the challenge description [4] as well as the further work of the named groups. For our classical solution the paper „Air traffic simulation in chemistry-climate model EMAC 2.41: AirTraf 1.0“ written by Hiroshi Yamashita et al.[6] acted as a basis. In the paper an aircraft routing module, which optimizes flight path with respect to non-CO2 effects, is suggested. The optimization with respect to the climate cost function (CCF) is handled as a single-objective minimization problem. It can be solved by an aircraft routing module, which comprises the Genetic Algorithm (GA) and finds an optimal flight trajectory including altitude changes.

The flowchart in figure 4 shows the working principle of our classical algorithm. It is as follows:

1. Firstly, the straight line solution is computed. It gives us the search bounds for the control points. In the cited paper a great-circle trajectory is used. However, this is not necessary in our case since we neglect the curvature of the earth.
2. The straight line solution is broken into 4 sections horizontally and

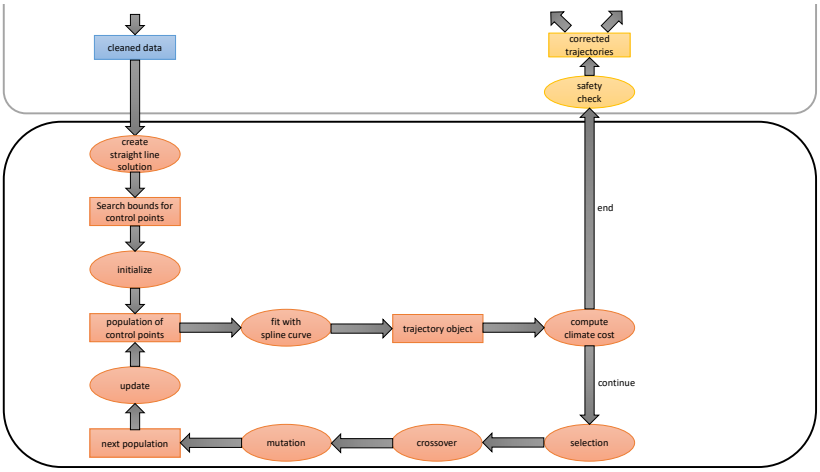

Figure 4: Flowchart of our classical solution routine.

five sections vertically. Parameters are randomly generated around the points dividing the sections, within the search bounds. These are our initial guesses for the control point parameters. For these guesses we call the fitness function for the first time.

3. Our fitness function takes a list of 11 parameters for the control points (3 are for the longitudinal coordinate, 3 for the latitudinal and 5 for the flight levels) as an input. The fitness function then obtains a trajectory by calling „curve_3D_solution(flight_nr, ctrl_pts)“, which fits a B-spline curve. From the trajectory the climate cost is computed and returned.
4. Now genetic operations (selection, crossover and mutation) are applied. With the new values the fitness function is called again.
5. We set the default parameters that the genetic algorithm will end after 50 iterations without improvement or after a maximum of 100 iterations. Of course they can be adjusted if wished.
6. If the genetic algorithm finishes, a report showing improvement of cost with iteration, the optimization result (the best list of parameters together with cost) and the optimal trajectory object are returned.

One problem, which occurred while running the genetic algorithm, is that the trajectory moved outside our defined area and with that into an area, where we have no atmospheric data. Since in real life we would take the whole world into account we allow the trajectories to move outside the area in the horizontal plane. We mapped the horizontal coordinates, which are outside our defined area, on the nearest available atmospheric data. We think this is legit since the longer way already results in a higher fuel consumption

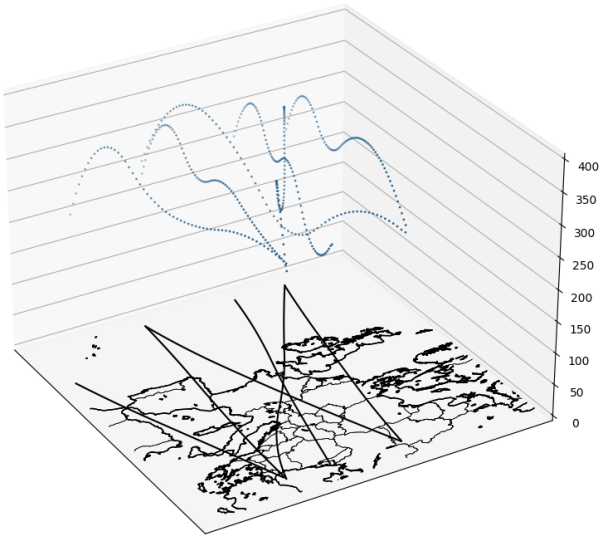

Figure 5: Flight trajectories of the flights 12, 34, 42, 57, 69, 81 calculated with the genetic algorithm. On the map the projection of the trajectories is shown.

and with that in an increased climate impact. Nevertheless, we saw a need to restrict the flight levels since in the real world a plane can not flight too close to the ground and also not too high. To prohibit this case, we defined a function „correct_for_boundaries“, which corrects the parts of the trajectory, which go outside our area of interest by replacing those parts through straight line solutions between the two point i and j if i+1 and j-1 are outside of flight level range. This solution is of course not optimal since it exists the possibility that on the straight line voxels with bad climate effect are crossed. However, we can neglect this effect for benchmarking, because it effects the classical and quantum solution in the same way.

By running our classical optimization algorithm, we find out the optimal trajectories. Csv files containing the trajectory for each flight and the corresponding climate cost can be find in our GitHub in the folder quantimize/output: `https://github.com/fjelljenta/Quantum-Challenge/tree/main/quantimize/output`. As an example the trajectory of flight 5 can be also found in the appendix. Moreover, you can find a visualization of the flight paths of the flights 12, 34, 42, 57, 69, 81 in figure 5. For clearness we only pictured five trajectories here.

To summarize, the genetic algorithm is a stochastic optimization algorithm suitable for continuous variables. It works well for optimizing the trajectory of a single flight. One limitation of genetic algorithm is that it does not take constraints given by safety regulations into account. Therefore, to obtain a

full solution we need to follow the routine described in section 5, which is to wrap the genetic algorithm in a for-loop, run from flight 0 to 99 one by one and perform the safety check in the end. After that we need to replace the conflicting trajectories by the next-best options and check again if the conflict is solved.

7 Our quantum solutions

The classical algorithm works solid. However, we thought a quantum algorithm might bring improvement. Therefore, we looked for candidate quantum algorithms by considering the following questions:
1. Is there a quantum algorithm that resembles the genetic algorithm, but outperforms it?
2. What are the most promising NISQ algorithms and which of them deal with the type of optimization problem we have?
By answering question one, we have found the quantum genetic algorithm. By answering question two, we found candidates such as quantum neural network (QNN) and QAOA/quantum annealing for solving QUBO problems.

In our view the QUBO solution is the most promising one. Nevertheless, we also looked at the other options and found promising features and disadvantages about them.

7.1 Quantum Solution 1: QUBO

We did not find any available NISQ algorithms directly adaptable to our constrained continuous variable optimization problem. Therefore, we believed it makes sense to start implementing a quantum algorithm with discrete variables. We found a way to simplify and reformulate our problem as a QUBO problem, or equivalently ground-state finding problem for an Ising Hamiltonian, which is commonly solved by QAOA algorithm on gate-based quantum computers (or correspondingly quantum annealing on D-wave hardware). In the last years, QAOA became a promising candidate to show quantum supremacy [7]. However, there are no examples yet which prove that there is a quantum advantage using QAOA on an optimization problem [8]. Nevertheless, one needs to state that the performance of QAOA depends on the performance of the classical optimization routine used to optimize the variational parameters. A lot of research is going on at the moment in order to find the perfect parameters. Once progress has made there it seems promising that QAOA could outperform classical routines.

An overview over this solution is presented in the flowchart in figure 6. We started by adopting some simplifications to the given model to run a scaled-down sample problem on a four qubit device. After that, it is necessary to design a way to scale up to be able to solve the real-world problems.

One simplification we applied for the proof-of-concept experiment is that we considered a 2D problem instead of 3D. That means we keep the flight

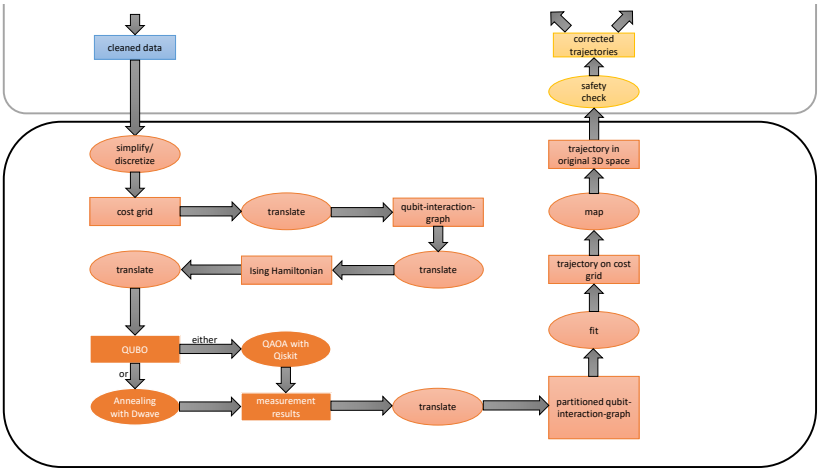

Figure 6: Flowchart of our quantum solution routine.

level fixed during the flight. However, we later show that the problem can be extended to 3D.

We came up with a creative way to map the original problem to a problem of finding the ground state of a two-dimensional rectangular lattice of qubits. The non-CO2 and CO2 emissions are mapped to the magnitudes of the interaction terms and to the single qubit terms, or to say it in graph language: to the values of vertices and edges. The starting point and ending point of a flight are placed at the opposite corners of the lattice, being fixed in state $|+\rangle$. The remaining qubits at the boundary are then divided into two chains, one in state $|0\rangle$ and the other in state $|1\rangle$. The centre qubits are all variable and initialized in state $|+\rangle$.

With this initial setup, it is guaranteed that at the end of the algorithm, the ground state of the lattice will always be a clean partition into two connected regions, one with qubits all in state $|0\rangle$ and the other all in state $|1\rangle$. Intuitively, we are making a cut such that the total cost of the broken edges is minimized, which, when mapped back to the original problem, means a trajectory with minimal climate cost.

This ground state finding problem is then changed to a QUBO problem, which is well known to be solvable on both quantum annealers and gate-based quantum computers via QAOA.

Our proof of principle experiment can be found under `https://github.com/fjelljenta/Quantum-Challenge/blob/main/quantimize/quantum/quantum_solution.py`.

We are convinced that this quantum algorithm is the most promising, because

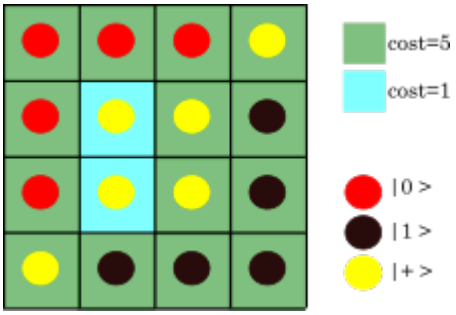

Figure 7: We divided the space in voxels with different atmospheric costs (here 1 and 5). In the middle of each voxel we place a qubit.

with a careful design of mapping the available data onto a cost grid, and by cutting the cost grid into fine-enough voxels, one can perfectly recover the original continuous optimization problem for a single flight, just as the GA does it. Our proof-of-concept experiment documented that since it returns the optimal solution, which was verified by performing a brute force check.

7.1.1 The proof-of-principle experiment

For our scaled-down example, we assume that the cost grid is given. More details regarding the derivation of cost grid from real data will be discussed in the next section. We consider here is a 4x4 cost grid with different values, which correspond to the atmospheric data plus a compensation for the travel direction (grid points deviating more from the straight line solution lead to higher travelling time and hence CO_2 emission). The cost grid takes two different values 1 and 5. In the middle of the voxels (In 2D they are actually pixels, but we will keep the name voxel for consistency) we place the qubits. This is visualized in figure 7.

Then we connect the centres of the voxels to form a 4x4 qubit lattice which we call as Q. We use fixed qubits ($|0\rangle$ and $|1\rangle$ states) at the boundaries, except that the two qubits representing the start and end of flight are set to $|+\rangle$ states. The remaining four qubits in the middle (q1, q2, q3, q4) are variable and initialized in the plus state by performing a Hadamard transformation. The qubit lattice is visualized in figure 8.

Each edge simply takes the average value of the two voxels the connected qubits are located. For example, the edge between q1 and the qubit above it takes value $(1+5)/2 = 3$. The full Hamiltonian of the lattice can be easily found by reading from the figure. We can just look at the Hilbert space formed by the 4 variable qubits. The interactions between two variable qubits

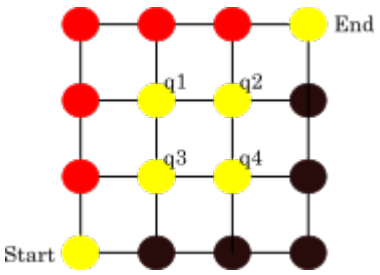

Figure 8: The qubits placed in the middle of the voxels as seen in figure 7 form a qubit grid. The edge between two qubits has a cost value, which is the average of the two voxels it connects. The boundary qubits are fixed to zero (red) or one (black). The start and end qubits (fixed), as well as the qubits in the middle of the grid (variable) are in the plus state (yellow).

give ZZ terms. The interactions between a fixed qubit and a variable qubit contribute to single-qubit Z terms. The interactions between two fixed qubits are constants and hence dropped.

$$H_C = -(\sum_{<i,j>} w_{ij} Z_i Z_j + \sum_{i=1}^{4} a_i Z_i)$$

Now the problem is formulated as an Ising Hamiltonian problem, which is equivalent to a QUBO problem by change of variable, and solvable by quantum annealers. We chose to take one step further and use the "qiskit_optimization" Python package by IBM to solve the QUBO problem by QAOA. The mixing Hamiltonian is standard: $H_M = \sum_{i=1}^{4} X_i$. The cost Hamiltonian is H_C above.

Due to high queuing time for real devices, we solved the problem with Qiskit's qasm simulator and the result was $|q_1 q_2 q_3 q_4\rangle = |0111\rangle$ giving cost of -20. We performed a brute force check and confirmed that it is indeed the optimal solution among the total $2^4 = 16$ candidates. With this outcome now the lattice Q is divided into two regions, one with all $|0\rangle$ qubits, the other with all $|1\rangle$ qubits. The optimal flight trajectory is now given by the border between the two regions. This can be seen in Figure 9.

7.1.2 A way to scale up

In this section, we'll first discuss the 2D case in more detail and then see how we can extend our solution to 3D.

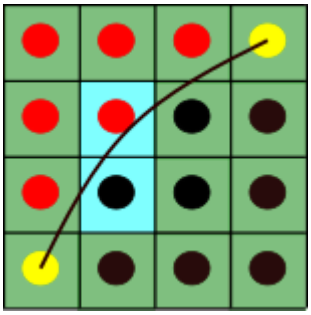

Figure 9: By minimizing the cost Hamiltonian we find the optimal state for the variable qubits. Our optimal trajectory is now the curve, which best separates the two regions. It is shown here in black.

So, in our problem, we were given a function we'll call c that returns the merged climate impact for a given set of location, flight level and time. For a single flight constrained to a 2D plane, we take this to be a function only of location $c = c(x, y)$, and because we're constrained to a single altitude speed and fuel burned per time can be taken constant. In fact we'll set this constant to be 1 to keep our math simple. Our cost function for a 2D trajectory Γ then comes out to be -

$$C(\Gamma) = \int_{\Gamma} c(x, y)\, dl$$

Our first step then is to discretize the space and turn this integral into a sum. We will represent our space by a lattice and assign cost $c_{i,j} = c(x, y)$ to the edges of neighboring points $i = (x_1, y_1)$ and $j = (x_2, y_2)$ where $x = \frac{x_1+x_2}{2}$ and $y = \frac{y_1+y_2}{2}$.
For any trajectory drawn in this lattice, let $E(\Gamma)$ be the set of edges that this trajectory intersects. The cost then is approximated by

$$C(\Gamma) = \sum_{(i,j)\in E(\Gamma)} c_{i,j} \cdot \Delta L$$

where ΔL is the distance travelled between consecutive edges. Here we've assumed that this distance is constant for every pair of consecutive edges. In our example we ensure this by picking a square lattice and assuming that our trajectory always cuts edges through the center. We also assume that when the trajectory enters a lattice cell or voxel, it moves to the voxel center in a straight line, and moves from there to the next edge in a straight line as well. In our example then, ΔL is the length of the side of a square in this grid.

(Note: The cost we're calculating is for an approximation of our trajectory that passes through the centers of our voxels, which may be longer than the real trajectory. This means that for certain trajectories, this sum can overshoot the actual cost. This error can be minimized using by triangular voxels, in which case, the calculated cost is at most $\frac{2}{\sqrt{3}} = 1.1547$ times the actual trajectory's cost. We used a square lattice however for the sake of simplicity, and because this decision bears no relevance when we generalize to the 3D case.)

Next, we remove parts of the lattice such that the start and end points of our trajectory are on opposite edges. Given that longer paths in general lead to higher costs, we think it's fair to assume that the true optimal trajectory won't lie outside these edges.
Now for every point i belonging to the lattice, let $x(i)$ be a binary value assigned to that point. The endpoints of our trajectory divide the lattice boundary points into two groups. Let the points in one part of the boundary be assigned the value 1 and assign 0 to the points on the rest of the boundary. Let $\vec{x}$ be the vector containing $x(i)$ for all points in the grid that are not on the boundary and letS $S(\vec{x}) = \{(i,j)|x(i) \neq x(j)\}$ where i and j are neighbouring points on the grid.
Define a lattice cost function C' as follows.

$$C'(\vec{x}) = \sum_{(i,j)\in S(\vec{x})} c_{i,j} \cdot \Delta L$$

For a given $\vec{x'}$, there exists at least one trajectory Γ' such that $E(\Gamma')$ is a subset of $S(\vec{x'})$ where $E(\Gamma')$ is the set of edges that Γ' intersects. To see this, consider the points on the boundary labelled 0. Group these points together with every neighboring point that's labelled 0, and call it G. Keep adding neighbor points labelled 0 to G till there are no points in G that have neighboring 0 points that aren't already in G. It is visually evident that this has to form a border with points labelled 1 and that this border shares the same end points as our trajectory.

Γ', the trajectory that we got from $\vec{x'}$, divides the grid into two regions like any valid trajectory would. Let $\vec{x'_o}$ represent the configuration where the points in these two regions are assigned the same label as the boundary points in those respective regions. $S(\vec{x'_o})$ is now just equal to $E(\Gamma')$, which means that $S(\vec{x'_o})$ is a subset of $S(\vec{x'})$. Assuming $c(x,y)$ is always positive, we get

$$C(\Gamma') = C'(\vec{x'_o}) < C'(\vec{x'})$$

This means that finding the minimum value of C is equivalent to finding the

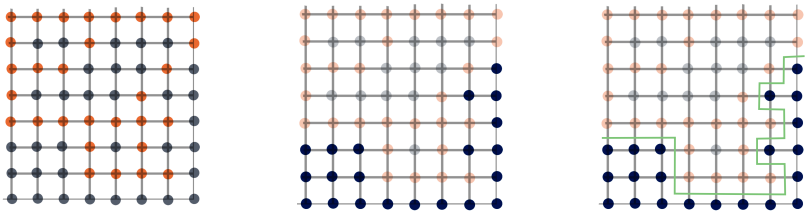

Figure 10: Here, the blue lattice points are the ones labelled 0 and the red points are those labelled 1

minimum value of C', and the optimum $\vec{x}$ corresponds to the lattice being split into two regions. The centers of the edges at the border between these regions can be connected together along with the start and end points to get the the trajectory with the minimum cost. Finding the optimum $\vec{x}$ is therefore equivalent to finding the optimum trajectory.
If we rework this cost function C' into a more convenient form, we get

$$C'(\vec{x}) = \sum_{\text{i,j are neighbors}} x(i) \cdot (1 - x(j)) \cdot c_{i,j} \cdot \Delta L$$

This is a quadratic function in binary variables that we need to minimize. It's a QUBO problem and it can be solved using QAOA where the binary variables in $\vec{x}$ are realized by qubits.

Now we generalize this to the 3D case. In 3D, we'd need to use 3 level qubits or 'qutrits' to apply this method. We'll use the same principle. We'll have a 3D lattice representing the whole space, where each lattice point can be assigned either 0, 1, or 2. We'll label the points on the surface such that there are three separated regions corresponding to each label, and the two faces where all three regions meet will be the start and end point of the trajectory.
We assign a positive cost to every face and add the costs of those faces that contain vertices of all three labels. Call these faces 'active faces'. We can then prove that minimizing this lattice cost will give us a single trail of active faces whose centres can be joined to form a trajectory. Then we need to set the cost that is to be assigned to each face in such a way that the overall lattice cost is approximately equal to the actual cost of the output trajectory.

So our first step would be to show that for any random assignment of labels for the inner lattice points, there will be at least one trajectory from the given start and end points that only cuts active faces. To show that, we first need to take a cross section and show that there always exists a single lattice point

with neighbors of all three colors. This is similar to what we did in the 2D case. We take two labels, say 0 and 1, and we consider the border points that are assigned those labels.
For each label, we create a group of points consisting of the boundary points, their neighbors of the same label, and their neighbors of the same label, and so on. It is visually evident that there will be one lattice point that neighbors points from both groups but is itself in neither group. It follows logically that this point must have label 2 and the face that it shares with the other two points must be an active face.

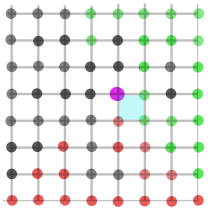

Figure 11: Here, the red and green lattice points are those points labelled 0 and 1 that are connected to the boundary of the lattice. The pink vertex is in neither group but is a neighbor to both groups and must be labelled 2. The highlighted square then is then an active face.

Also, notice that when we do this for all three pairs of labels, we'll get three active faces in a single cross section. The lattice cost for a single cross section however can be made minimum if there's only one active face, i.e., if all three groups associated with the boundary coincide at a single face and there are no other active faces. If we put two such cross sections on top of each other, we can see that a trail of active faces is formed between the two active faces belonging to each cross section. This roughly shows that the allotment of labels that minimizes the lattice cost is always associated with a trajectory.

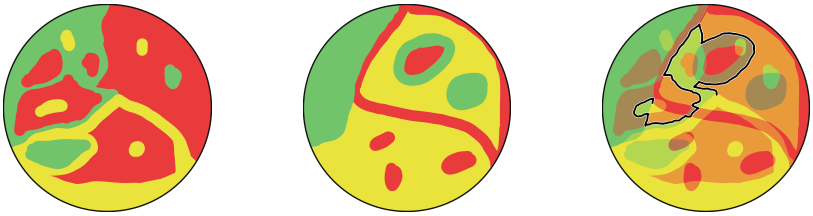

Figure 12: Let the first two images be two consecutive cross sections of a very dense lattice, each of which contain only one active face. The red areas contain those points that are labelled 0; green, 1; and yellow, 2. If you placed the two cross sections on top of each other and now marked every point that touches all three colors, you get a continuous path.

Our next step would be to assign costs to each face such that the minimum lattice cost is approximately equal to the cost of the trajectory associated with the corresponding allotment of labels. To start, we first need to recall what our trajectory's cost function is. In our problem, the trajectory can be divided into smaller sub-trajectories where the airplane is either cruising, climbing, or descending. The cost of each subtrajectory then is given by

$$C(\Gamma_i) = \int_{\Gamma} c(x,y,z) \cdot F \, dl$$

where $c(x,y,z)$ is the cost of burning fuel at a certain location (not dependent on time since we're considering only a single flight) and F is the fuel burned per distance travelled and depends on whether the airplane is cruising, climbing or descending in this part of the trajectory. To differentiate between these three cases, one thing we can do is we could pick a lattice consisting of triangular prisms for unit cells.
Consider a general trajectory that makes a straight line in the x-y plane. Set the triangular faces of our lattice to be parallel to this trajectory, and set the three quadrilateral faces of the prisms to be aligned parallel to the slopes of the cruising, climbing, and descending parts of such trajectory. Let's call these faces 'cruise faces', 'climb faces' and 'descent faces' accordingly.

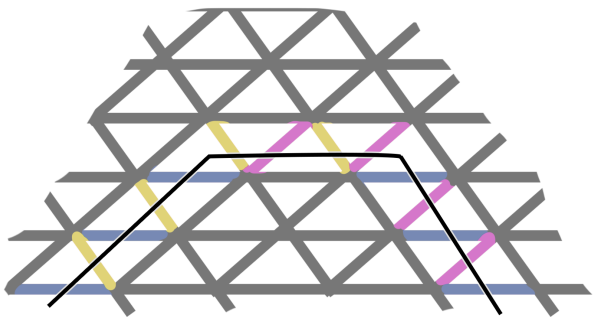

Figure 13: Trajectory through side cross section of prism lattice.

We can see from the figure that while cruising, the trajectory intersects the climb and descent faces; while climbing it intersects the cruise and descent faces; and while descending it intersects the cruise and climb faces. So we

know that the sum of costs of a climb face and a descent face should be the cost of a trajectory cruising from the center of one upright prism to the center of the upright prism in front of it (we take this distance to be ΔL).

$$c'_{climb}(x_1,y_1,z_1)+c'_{descent}(x_2,y_2,z_2)=[c(x_1,y_1,z_1)+c(x_2,y_2,z_2)]\cdot\Delta L\cdot F_{cruise}$$

We can form three such equations and solve them to get the cost associated with every quadrilateral face. The cost of the triangle face can be taken as

$$c'_{triangle}(x,y,z)=c(x,y,z)\cdot\Delta L\cdot F_{cruise}$$

Here ΔL is the distance between two consecutive triangular faces.

So now we have the cost associated with any face, which we will call $c'(x,y,z)$, where (x,y,z) is the position of the face center.

The total lattice cost is now approximately the actual cost of the trajectory. Just like the 2D case, the reason for the cost being slightly off is because the lattice is biased toward certain directions. In this case, for example, we've assumed that the flight only climbs/descends when moving along a certain direction. We can tame this issue by setting that direction to be the direction of the straight line trajectory.

Now, similar to the 2D case, we have qutrits representing every point in our lattice other than the boundary points.

So, if the state of a qutrit were represented like so

$$|\psi\rangle=\alpha|0\rangle+\beta|1\rangle+\gamma|2\rangle=\begin{bmatrix}\alpha\\ \beta\\ \gamma\end{bmatrix}$$

then we can construct a Cost Hamiltonian

$$H_C=\frac{1}{2}\sum_{(q_1,q_2,q_3,q_4)_\square}c'(x,y,z)\cdot I_{q_1}\otimes(\frac{I+P}{2})_{q_2}\otimes(\frac{I+Q}{2})_{q_3}\otimes(\frac{I+R}{2})_{q_4}$$
$$+\sum_{(q_1,q_2,q_3)_\Delta}(\frac{I+P}{2})_{q_1}\otimes(\frac{I+Q}{2})_{q_2}\otimes(\frac{I+R}{2})_{q_3}$$

where the first summation is over all quadrilateral lattice faces $(q_1,q_2,q_3,q_4)_\square$ and all their permutations, and the second summation is over all triangular

lattice faces $(q_1, q_2, q_3)_\Delta$ and their permutations. (x, y, z) is the position of the center of the face being summed over. I is the identity matrix and

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

$$Q = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

$$R = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

We can solve this problem using QAOA, which in principle should work on a qutrit system as well. The cost hamiltonian will be a diagonal matrix which should be convenient to work with. The mixer hamiltonian can be any standard hamiltonian that doesn't commute with the cost hamiltonian.

Each qutrit will encode the label given to each lattice point (other than the ones on the boundary of the lattice). So the number of qutrits required will depend on how we discretize the space. We are given atmospheric data for 6076 voxels in a cubic lattice though we could definitely afford to make this domain coarser when performing our algorithm, plus there are many voxels we won't be using depending on the flight start and end points. We can approximately say that we would need between 500 and 3000 qutrits to run this algorithm, depending on the level of accuracy we require. The depth for 1 layer of evolution along the cost Hamiltonian should be around 324 (number of terms in the cost hamiltonian that involve a specific qutrit) regardless of the number of qutrits. Ideally, that of the mixer Hamiltonian would be 1. Based on what's typically seen in QAOA problems, the number of layer repetitions (p) to get a reasonable solution increases with number of qubits. T_1 time of qutrits today is on average $40\mu s$, and the time it takes to operate a two qutrit gate is around $768ns$ (Reference). This means that the maximum depth we can get to today before our device succumbs to noise is 52, whereas we need a depth of 325 for an ansatz of just 1 layer. So we still have a long way to go. If qutrits are not available, another approach worth mentioning is to simulate qutrits on a qubit device, though this would be a waste of resources. One way to do this is to have a qutrit represented by two qubits where $|0\rangle$ will be $|01\rangle$, $|1\rangle$ will be $|10\rangle$, and $|2\rangle$ will be $|11\rangle$. P can then be replaced by Z acting on the second qubit, Q with Z acting on the first qubit, and R with Z acting on both qubits. The initial state and the mixer hamiltonian however

will need to be picked carefully so as to make sure all the qubit pairs avoid the $|00\rangle$ state since that state doesn't represent anything.

7.2 Quantum inspired Solution 2: QGA

Another possible quantum solution is the quantum-inspired quantum genetic algorithm (QGA), which is a low hanging fruit after we used the genetic algorithm as a classical solution.

The QGA is the combination of quantum computation and the genetic algorithm. It is a heuristic optimization method based on simulated genetic mechanisms, such as mutations and crossovers, and on population dynamical processes, such as reproduction and selection. In general heuristic algorithms are simple to apply as they typically consider the problem as a black box and only access it via a problem-dependent fitness function. [9]

By looking into this kind of algorithm, we found a GitHub repository, which provides python code for the QGA [10]. This code was used for a maximization problem, but we managed to adopt the code and map it on our problem. Our QGA implementation can be find under `https://github.com/fjelljenta/Quantum-Challenge/blob/main/quantimize/quantum/QGA.py`. In the Quantimize.ipynb the QGA is applied to the flight data and also benchmarked against the other algorithms.

Running the QGA returns us the flight trajectories, from which the costs can be computed. Csv files containing the trajectories and the corresponding climate costs for all flights can be found in our GitHub under `https://github.com/fjelljenta/Quantum-Challenge/tree/main/quantimize/output`. One example trajectory of flight 5 solved by the QGA can also be found in the appendix. Moreover, you can find a visualization of the flight paths of flights 12, 34, 42, 57, 69 and 81 in figure 14. We found out, that our QGA algorithm outperforms the straight line solution. However, it is weaker than the classical genetic algorithm and it also do not provide a speedup. For a more detailed benchmarking see section 8. A further limitation is, that it relies on a look up table for updating rotation angles.

7.3 Quantum Solution 3: QNN

Quantum Neural Network (QNN), also known as Variational Quantum Circuit (VQC) is a class of NISQ-era quantum-hybrid algorithms for optimization

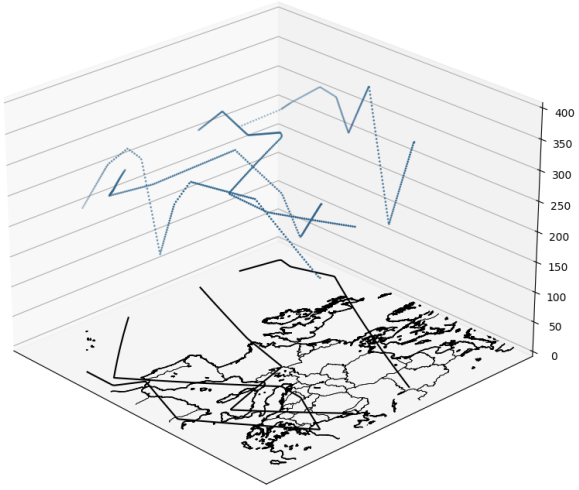

Figure 14: Flight trajectories of the flights 12, 34, 42, 57, 69, 81 calculated with the genetic algorithm. On the map the projection of the trajectories is shown.

and machine learning under active research. As QNN for binary classification has been widely studied [11], it is tempting to use QNN for detecting conflicts between trajectories and thus perform safety check, which could potentially be more efficient. However, a balance between accuracy and efficiency needs to be carefully evaluated.

Another potential usage of QNN, which we studied in more details, is to encode flight information (starting time and position and the destination) into qubits, train parametrized gates to learn the atmospheric data, and output quantum states that with some additional post-processing give the optimal trajectory. We implemented an example QNN using Pennylane's framework for demonstration purpose. We used "amplitude embedding " and "StrongEntanglingLayers" as circuit Ansätze. The training is done by simultaneous perturbation stochastic approximation (SPSA). The choices of Ansätze, classical optimizer and hyperparameters could be improved. However, it is likely that the large data and complicated mathematical model we have require either a big breadth (number of qubits) or a big depth (layers of quantum gates, which leads to the accumulation of errors), making it challenging to be constructed or trained on currently available devices. However, as more research is being done, we do believe that in the future, QNN could be useful for the problem if used wisely.

7.4 Quantum Solution 4: Graphic solution

A last idea for a quantum algorithm was a graphic solution. However, to realize this fault tolerance quantum computers will be necessary. Due to the limited time of the challenge, we decided to not go further into this idea. Since the other solution can be relevant before fault tolerance computing is even possible. Nevertheless, we wanted to shortly explain this idea in the appendix since we think it could be promising to look into it again once fault tolerance quantum computing is invented.

8 Benchmarking and discussion

To verify our code, we benchmarked the straight line solution, the GA and the QGA. We see the straight line solution as the state of the art trajectory since airline companies choose the most direct trajectories with the lowest fuel consumption and hence lowest CO2 emission, but not necessarily the lowest combined effect on climate. Therefore, our goal is to show that our solutions outperform the state of the art solution with respect to climate impact, but also by taking other important parameters into account. To show that we benchmarked with respect to four criteria: the climate impact of the optimal solutions, the computational time needed, the flight time and the fuel consumption. Especially, the last two criteria are important for the companies, because they come along with costs and customer satisfaction.

Since the straight line solution is a fixed solutions in a sense that it will always produce the same trajectory for the same start and ending parameters, it is not possible to resolve the crashing problems by running the algorithm for the straight line solution again. Therefore, we created a benchmark of all three mentioned algorithms neglecting the safety conditions as well as a benchmark of the genetic algorithm vs the quantum genetic algorithm including the air safety protocol.

The results of our benchmarks are shown in the figures 15, 16, 17 and 18 where the left part always shows the benchmarking with neglected air safety check and the right part shows the benchmarking including the air safety protocol. As one can see in direct comparison of left and right part, the climate impact of the optimal solutions, the flight times and the fuel consumptions are almost the same with and without the air safety check, just the computational time needed is approximately three times higher for the benchmark including the air security. Therefore, we do not need to distinguish further if we run the airsafety protocol for the benchmark or not.

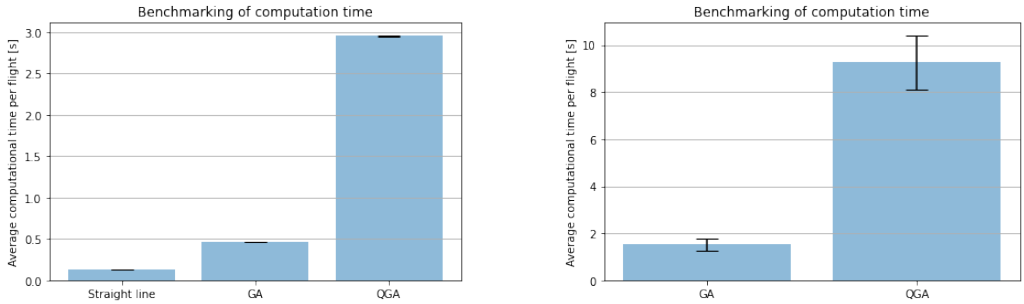

Figure 15: Benchmarking of the average computational time per flight. On the left the staight line solution, the genetic algorithm and the quantum genetic algorithm are benchmarked without checking the safety regulations. On the right side the genetic algorithm and the quantum genetic algorithm are benchmarked while fulfilling the safety protocol. We took five runs each over all flights to produce these results.

We found out that, the straight line solution is the fastest solution to compute, but also has the worst climate impact and a similar high fuel consumption like the quantum genetic algorithm. We were a little bit surprised that it did not offer the fastest flight times, although it has the shortest distance. However, by having a closer look, it makes sense since the flight speed depends on the flight level and the straight line solution stays at one level, which can be slower than the others.

Both the classical and the quantum genetic algorithm outperforms the straight line solution with respect to the climate impact. The flight times for the GA and the straight line solution are very close together. The QGA is a little bit slower, but its extra time needed is below half an hour. Therefore, customer satisfaction is not impacted by reducing the climate impact of flying.

While the fuel consumption for the straight line solution and the QGA is very similar, the fuel consumption for the GA is about 15 % lower than the one for the other both algorithms, but we are confident that with improving the QGA further, it will come to similar results like the GA. A reduced fuel amount per flight is in the interest of the airline companies as well as the passengers who have to pay less for the flight and therefore not only the environment profits from such optimized flight trajectories.

We did not include the QNN in our benchmarking since the result is very noisy. Our code regarding the QNN should serve more as inspiration that quantum neural networks will have a chance to perform outstandingly if devices with more qubits become state of the art. At the moment, however, this approach has been beaten by the other quantum solutions.

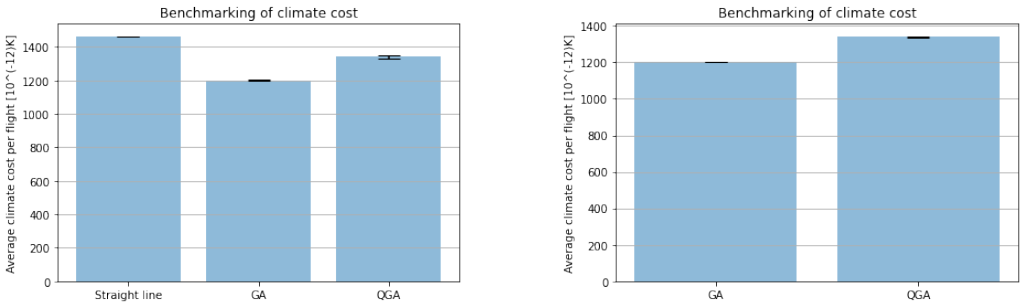

Figure 16: Benchmarking of the average climate impact per flight. On the left the staight line solution, the genetic algorithm and the quantum genetic algorithm are benchmarked without checking the safety regulations. On the right side the genetic algorithm and the quantum genetic algorithm are benchmarked while fulfilling the safety protocol. We took five runs each over all flights to produce these results.

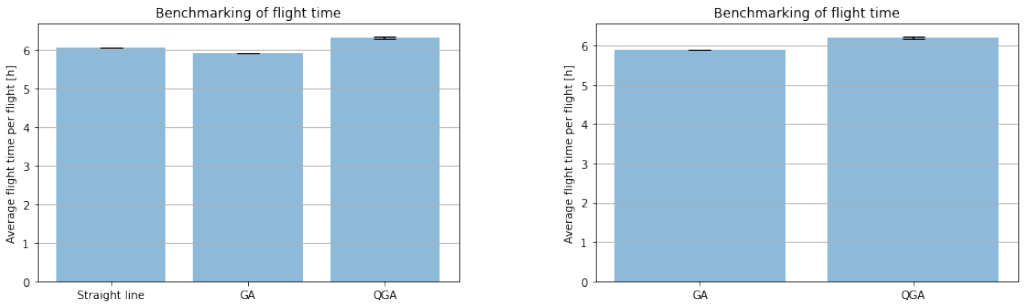

Figure 17: Benchmarking of the average flight time per flight. On the left the staight line solution, the genetic algorithm and the quantum genetic algorithm are benchmarked without checking the safety regulations. On the right side the genetic algorithm and the quantum genetic algorithm are benchmarked while fulfilling the safety protocol. We took five runs each over all flights to produce these results.

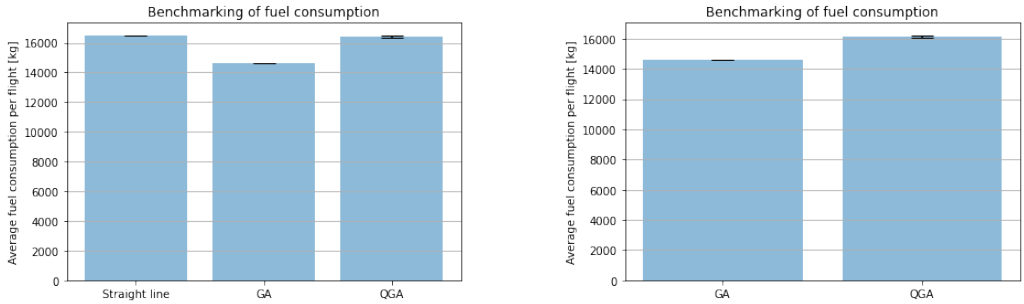

Figure 18: Benchmarking of the average fuel consumption per flight. On the left the staight line solution, the genetic algorithm and the quantum genetic algorithm are benchmarked without checking the safety regulations. On the right side the genetic algorithm and the quantum genetic algorithm are benchmarked while fulfilling the safety protocol. We took five runs each over all flights to produce these results

Moreover, it is hard to benchmark our QUBO solution against the others. In our proof of principle, it delivers the true optimum (find by brute force), which is exactly what we want. Nevertheless, to have a good result we would need to have a finer grid of qubits to make the problem „more continuous“ since it is discrete at the moment.

9 Conclusion and outlook

We think, that at the moment the classical genetic algorithm is the best choice for a big data set. Therefore, if airlines want to optimize their flight trajectories in order to reduce their climate impact, we recommend to use the genetic algorithm until larger and better NISQ hardware is available. From our point of view the most promising quantum algorithm we found is to discretize the space, transform the cost function into a QUBO form and then solve it by quantum annealing or the QAOA algorithm on gate-based quantum computers such as IBM's superconducting quantum devices.

A four qubit device was enough to building a scaled-down problem for demonstration purpose. However, more qubits, in the order of 1000, are needed to run it on the big data set without too much truncation.

The quantum genetic algorithm performs solid, but does not give an obvious advantage over the classical genetic algorithms. With more research, especially on updating the rotation angle, it may be possible that it slightly outperforms the GA. However, we do not think, that it will give the hoped-for

big quantum advantage.

The quantum neural network, we implemented, is not trainable with the currently available number of qubits. However, we do believe that this is a promising concept, which can be useful in the future, when larger devices are standard.

10 Appendix

Quantum algorithm for short path search in a directed graph:
In order to use this method, we first need to construct a graph. For that we took the following assumptions:

- It is only possible to fly forwards, that means every step has a fixed change in x-direction. With that we make sure, that the graph is directed.
- The length of the connection between two knots is the value of the cost function (climate impact of the fight distance).
- In order to get these values we need to have to discrete values, which correspond to the knots. (First: maybe best to use corners of the atmospheric data voxels as knots for the graph. The more knots we use the more qubits we would need to implement our graph.
- Moreover, we need to check which points are reachable from a certain point, so that only those knots get a connection which are allowed to fly in between.
- For the end point we have only given latitudinal and longitudinal point and flight level can vary. We need to introduce an extra point with cost zero (which would also provide the option to include a landing cost) to have a fixed end point.
- In a first step the graph would be the same for all flights. However, one could also think about directly performing a security check after each flight and making path which would violate the safety check more costly for the next plane.

After implementing the graph, we would need to find the shortest path through our graph for ever plane. We found one paper, which suggested an algorithm for that: `https://link.springer.com/content/pdf/10.3103/S0278641919010023.pdf`. The paper suggests to use see the different paths with different costs as an unstructured database and uses Grover's algorithm to search for the entry with the minimal climate cost.

Although we liked this idea we decided against it since it seems not very useful with the few accessible qubits on nowadays devices.

Flight trajectory for flight 5 solved by the genetic algorithm:

Table 1: Trajectory of flight number 5 calculated with the genetic algorithm. The climate cost for this flight is $1075.22 \cdot 10^{-12}$ K.

Flight number	5	Cost	1075.21959
Long	Lat	Flight Level	Time
-16.00	34.000000	210.000000	07:15:00
-15.50	34.289142	225.285068	07:19:11
-15.00	34.581470	239.781342	07:23:15
-14.50	34.876984	253.488823	07:27:13
-14.00	35.175684	266.407510	07:31:11
-13.50	35.477571	278.537404	07:35:03
-13.00	35.782643	289.878505	07:38:56
-12.50	36.090902	300.430812	07:42:46
-12.00	36.402346	310.194326	07:46:39
-11.50	36.716977	319.169046	07:50:33
-11.00	37.034793	327.354973	07:54:28
-10.50	37.355796	334.752107	07:58:26
-10.00	37.679985	341.360447	08:02:25
-9.50	38.007359	347.179994	08:06:25
-9.00	38.337920	352.210747	08:10:28
-8.50	38.671667	356.452707	08:14:32
-8.00	39.008600	359.905874	08:18:37
-7.50	39.348719	362.570247	08:22:43
-7.00	39.692024	364.445827	08:26:50
-6.50	40.038516	365.532613	08:30:58
-6.00	40.388193	365.830606	08:35:07
-5.50	40.741056	365.339805	08:39:17
-5.00	41.097106	364.060212	08:43:29
-4.50	41.456341	362.010866	08:47:42
-4.00	41.818763	359.820149	08:51:55
-3.50	42.184370	357.792727	08:56:10
-3.00	42.553164	355.928599	09:00:25
-2.50	42.925144	354.227765	09:04:42
-2.00	43.300310	352.690225	09:09:00
-1.50	43.678661	351.315981	09:13:18
-1.00	44.060199	350.105030	09:17:38
-0.50	44.443021	349.057374	09:21:56
0.00	44.823866	348.173013	09:26:13
0.50	45.202677	347.451946	09:30:30

1.00	45.579456	346.894173	09:34:46
1.50	45.954202	346.499695	09:39:01
2.00	46.326915	346.268511	09:43:16
2.50	46.697595	346.200621	09:47:30
3.00	47.066242	346.296027	09:51:44
3.50	47.432857	346.409406	09:55:57
4.00	47.797438	346.395442	10:00:09
4.50	48.159987	346.254134	10:04:20
5.00	48.520503	345.985483	10:08:30
5.50	48.878985	345.589488	10:12:40
6.00	49.235435	345.066150	10:16:50
6.50	49.589852	344.415468	10:20:59
7.00	49.942236	343.637442	10:25:07
7.50	50.292588	342.732073	10:29:14
8.00	50.640906	341.699360	10:33:21
8.50	50.987192	340.539304	10:37:27
9.00	51.332047	339.251904	10:41:32
9.50	51.676053	337.837161	10:45:37
10.00	52.019209	336.295074	10:49:42
10.50	52.361515	334.625643	10:53:47
11.00	52.702972	332.916710	10:57:51
11.50	53.043578	331.349445	11:01:55
12.00	53.383335	329.929337	11:05:57
12.50	53.722242	328.656386	11:09:58
13.00	54.060299	327.530592	11:13:59
13.50	54.397506	326.551955	11:18:00
14.00	54.733863	325.720475	11:22:01
14.50	55.069371	325.036152	11:26:01
15.00	55.404028	324.498986	11:30:01
15.50	55.737836	324.108977	11:34:01
16.00	56.070794	323.866126	11:38:01
16.50	56.402902	323.770431	11:42:00
17.00	56.734160	323.821894	11:46:00
17.50	57.064568	324.020514	11:49:59
18.00	57.394127	324.366290	11:53:58
18.50	57.722835	324.859224	11:57:56
19.00	58.050694	325.499315	12:01:54
19.50	58.377703	326.286563	12:05:52
20.00	58.703862	327.220968	12:09:49
20.50	59.029171	328.302531	12:13:46
21.00	59.353631	329.531250	12:17:43

21.50	59.677240	330.907126	12:21:42
22.00	60.000000	330.907126	12:25:41

Flight trajectory for flight 5 solved by the quantum genetic algorithm:

Table 2: Trajectory of flight number 5 calculated with the quantum genetic algorithm. The climate cost for this flight is $1152.12 \cdot 10^{-12}$ K

Flight_Nr	5	Cost	1152.1224156
Long	Lat	Flight_Level	Time
-16.00	34.000000	210.000000	07:15:00
-15.70	34.000000	212.763158	07:17:00
-15.40	34.000000	215.526316	07:19:00
-15.10	34.000000	218.289475	07:21:00
-14.80	34.000000	221.052633	07:23:00
-14.50	34.000000	223.815791	07:25:00
-14.20	34.000000	226.578949	07:27:00
-13.90	34.000000	229.342107	07:29:00
-13.60	34.000000	232.105265	07:30:55
-13.30	34.170866	234.868424	07:33:19
-13.00	34.377238	237.631582	07:35:54
-12.70	34.583609	240.394740	07:38:29
-12.40	34.789980	243.157898	07:41:04
-12.10	34.996351	245.921056	07:43:39
-11.80	35.202722	248.684214	07:46:14
-11.50	35.409094	251.447373	07:48:45
-11.20	35.615465	254.210531	07:51:16
-10.90	35.821836	256.973689	07:53:47
-10.60	36.028207	259.736847	07:56:18
-10.30	36.234578	262.500005	07:58:49
-10.00	36.440950	265.263163	08:01:20
-9.70	36.647321	268.026322	08:03:51
-9.40	36.853692	270.789480	08:06:17
-9.10	37.060063	273.552638	08:08:43
-8.80	37.266434	276.315796	08:11:09
-8.50	37.472806	279.078954	08:13:35
-8.20	37.679177	281.578954	08:16:02
-7.90	37.885548	283.947375	08:18:29
-7.60	38.091919	286.315796	08:20:56
-7.30	38.298290	288.684217	08:23:23
-7.00	38.504661	291.052639	08:25:48
-6.70	38.711033	293.421060	08:28:12
-6.40	38.917404	295.789481	08:30:36

-6.10	39.123775	298.157903	08:33:00
-5.80	39.330146	300.526324	08:35:24
-5.50	39.536517	302.894745	08:37:48
-5.20	39.742889	305.263166	08:40:12
-4.90	39.949260	307.631588	08:42:36
-4.60	40.155631	310.000009	08:45:01
-4.30	40.362002	312.368430	08:47:26
-4.00	40.568373	314.736852	08:49:51
-3.70	40.774745	317.105273	08:52:16
-3.40	40.981116	319.473694	08:54:41
-3.10	41.187487	321.842115	08:57:06
-2.80	41.393858	324.210537	08:59:31
-2.50	41.600229	326.578958	09:01:56
-2.20	41.806601	328.947379	09:04:21
-1.90	42.012972	331.315801	09:06:48
-1.60	42.219343	333.684222	09:09:15
-1.30	42.425714	336.052643	09:11:42
-1.00	42.632085	338.421064	09:14:09
-0.70	42.838456	339.473676	09:16:35
-0.40	43.044828	337.894729	09:19:01
-0.10	43.251199	336.315781	09:21:27
0.20	43.457570	334.736834	09:23:53
0.50	43.663941	333.157886	09:26:19
0.80	43.870312	331.578939	09:28:45
1.10	44.076684	329.999991	09:31:10
1.40	44.283055	328.421043	09:33:36
1.70	44.489426	326.842096	09:36:02
2.00	44.695797	325.263148	09:38:28
2.30	44.902168	323.684201	09:40:54
2.60	45.108540	322.105253	09:43:20
2.90	45.314911	320.526306	09:45:46
3.20	45.521282	318.947358	09:48:12
3.50	45.727653	317.368411	09:50:38
3.80	45.974035	315.789463	09:53:17
4.10	46.244561	314.210516	09:56:05
4.40	46.515088	312.631568	09:58:53
4.70	46.785614	311.052621	10:01:41
5.00	47.056140	309.473673	10:04:27
5.30	47.326667	307.894726	10:07:12
5.60	47.597193	306.315778	10:09:57
5.90	47.867719	304.736831	10:12:42

6.20	48.138246	303.157883	10:15:27
6.50	48.408772	301.578936	10:18:12
6.80	48.679298	300.000006	10:20:57
7.10	48.949825	300.789480	10:23:43
7.40	49.220351	301.578954	10:26:29
7.70	49.490877	302.368427	10:29:15
8.00	49.761404	303.157901	10:32:01
8.30	50.031930	303.947375	10:34:47
8.60	50.302456	304.736849	10:37:33
8.90	50.572982	305.526322	10:40:19
9.20	50.843509	306.315796	10:43:05
9.50	51.114035	307.105270	10:45:51
9.80	51.384561	307.894744	10:48:37
10.10	51.655088	308.684217	10:51:23
10.40	51.925614	309.473691	10:54:09
10.70	52.196140	310.263165	10:56:57
11.00	52.466667	311.052639	10:59:44
11.30	52.737193	311.842112	11:02:31
11.60	53.007719	312.631586	11:05:18
11.90	53.278246	313.421060	11:08:05
12.20	53.548772	314.210534	11:10:52
12.50	53.819298	315.000007	11:13:39
12.80	54.089825	315.789481	11:16:26
13.10	54.360351	316.578955	11:19:13
13.40	54.630877	317.368429	11:22:00
13.70	54.901404	318.157903	11:24:47
14.00	55.171930	318.947376	11:27:34
14.30	55.442456	319.736850	11:30:21
14.60	55.712982	318.947352	11:33:09
14.90	55.983509	317.368405	11:35:57
15.20	56.254035	315.789457	11:38:45
15.50	56.524561	314.210510	11:41:33
15.80	56.795088	312.631562	11:44:21
16.10	56.976454	311.052615	11:46:39
16.40	57.130194	309.473667	11:48:48
16.70	57.283934	307.894720	11:50:56
17.00	57.437673	306.315772	11:53:04
17.30	57.591413	304.736825	11:55:12
17.60	57.745152	303.157877	11:57:20
17.90	57.898892	301.578930	11:59:28
18.20	58.052632	299.999982	12:01:36

18.50	58.206371	298.421034	12:03:44
18.80	58.360111	296.842087	12:05:52
19.10	58.513850	295.263139	12:08:00
19.40	58.667590	293.684192	12:10:08
19.70	58.821330	292.105244	12:12:16
20.00	58.975069	290.526297	12:14:24
20.30	59.128809	288.947349	12:16:35
20.60	59.282548	287.368402	12:18:46
20.90	59.436288	285.789454	12:20:57
21.20	59.590028	284.210507	12:23:08
21.50	59.743767	282.631559	12:25:19
21.80	59.897507	281.052612	12:27:30
22.00	60.000000	281.052612	12:28:57

References

[1] UN. The climate crisis - a race we can win. https://www.un.org/en/un75/climate-crisis-race-we-can-win, 2020.

[2] Jeff Overton. Fact sheet | the growth in greenhouse gas emissions from commercial aviation. https://www.eesi.org/papers/view/fact-sheet-the-growth-in-greenhouse-gas-emissions-from-commercial-aviation#5, 2019.

[3] ROBERT Hackett. Ibm plans a huge leap in superfast quantum computing by 2023. https://fortune.com/2020/09/15/ibm-quantum-computer-1-million-qubits-by-2030/, 2020.

[4] Barbara Wellmann and Henrike von Zimmermann. Quantum climate challenge 2022 reducing the climate impact of air travel via optimized routing. *Deloitte*, 2022.

[5] Cadence CFD. An introduction to b-spline curves. https://resources.system-analysis.cadence.com/blog/msa2022-an-introduction-to-b-spline-curves, 2020.

[6] H. Yamashita, V. Grewe, P. Jöckel, F. Linke, M. Schaefer, , and D Sasaki. Air traffic simulation in chemistry-climate model. *EMAC 2.41: AirTraf 1.0, Geosci. Model Dev., 9, 3363–3392*, 2016. doi: https://doi.org/10.5194/gmd-9-3363-2016.

[7] Sami Khairy, Ruslan Shaydulin, Lukasz Cincio, Yuri Alexeev, and Prasanna Balaprakash. Reinforcement learning for quantum approximate optimization. https://sc19.supercomputing.org/proceedings/tech_poster/poster_files/rpost240s2-file3.pdf.

[8] Marwahaha. The living qaoa reference. https://marwahaha.github.io/qaoa-reference/, 2022.

[9] Ruben Ibarrondo Lopez. Quantum genetic algorithms. https://addi.ehu.es/bitstream/handle/10810/49102-/TFG_Ibarrondo_Lopez_Ruben.pdf, 2020.

[10] ResearchCodesHub. Quantumgeneticalgorithms. https://github.com/ResearchCodesHub/QuantumGeneticAlgorithms, 2016.

[11] Edward Farhi and Hartmut Neven. Classification with quantum neural networks on near term processors. https://arxiv.org/pdf/1802.06002.pdf.