Modeling Flight Trajectory

Zachary Kaster | Luke Nasby

Indiana University-Purdue University Indianapolis

Indianapolis, IN, USA

zckaster@iu.edu | lenasby@iu.edu

Introduction

Airline industries are expected to lose an estimated $84.3 billion dollars in the current year (2020).[4] As many airlines are looking for ways to cut costs in order to keep planes flying and attract as many passengers as possible with lower ticket prices, one way to cut those costs is to more efficiently route airplanes. This includes not only pre-flight trajectory calculations, but also during flight trajectory calculations. This will help reduce the amount of fuel used on each flight, which is a major contributor to the expenditures as fuel accounts for 15% of overall costs for the airlines industry.

By implementing a real time model for on board flight calculation, airplanes will not only save on fuel but will also reduce the amount of harmful pollutants released into the environment such as CO2, CO, nitrogen oxides, sulphur oxides and particulate matter. Previous attempts and successes include using reinforcement learning in order to affect how the plane flies.[2] In this model, an agent is trained in a simulated environment in order to choose the best path through weather and other obstacles during flight. This resulted in a shorter travel time, less fuel consumption and less pollutants being emitted into the air.

This is similar to the approach taken in this paper, where the plane will be in a simulated environment with different weather systems that force the proposed model to make corrections to its flight trajectory in real time. The constraints for this problem are that the agent must take the fewest steps through the 3D graph which represents its space. Using wind maps from different weather data providers, the graph will have different weights at each connection representing the cost of moving from one node to the next node. A genetic algorithm will be used to start off with a large population of agents which will improve the rate at which the AI improves. In order to reduce the state space and reduce time spent searching the space of possible moves there are only so many places a plane could be within the next step depending on its

velocity. Therefore, a reduced vision graph will be used to only consider spaces which are within the reach of the plane by the time it has moved one unit depending on its velocity. What should be seen from this experiment is a path which is arcing or avoiding some space of high wind resistance in order to reduce time. An idealized route would be straight from point A to point B. This model will take the optimal route instead.

The plan to implement the AI is to render a height map of a sample area of the globe and format it into a 3D array. At first there will be no turbulence and the AI will just learn to go from point A to point B. Then the weather patterns that cause turbulence will be introduced in the simulation in order to produce closer to real world results. These tests should result in precise cost analysis and routing.

In order to answer the question, how does routing affect commercial aircraft efficiency, we will be using reinforcement learning to reward and penalize the AI so it can determine the optimal path through weather forecasts and other conditions. The model will use height maps of the U.S. and weather forecasts for those areas. The objectives of the agent is to use the least fuel possible and get to the destination as fast as possible. The end product will be an agent trained on a genetic algorithm which picks the optimal route every time through varying conditions.

Related Works

In [2], the authors propose an efficient algorithm for finding the optimal path within a 3-D space which is cluttered with complex polygons that pose as obstacles between the AI and the goal. The defined constraints are that the algorithm must take the shortest path while also being the most energy efficient from polygon to polygon. In order to assess the cost from one point to another point, an extension of the visibility graph(VG) approach is used. Since their space was 3-Dimensional, the new graph reflects that space. The algorithm will choose the sub-shortest path with the given edges found, assuming it is within the visible range. Otherwise, it will move to the next closest visible point closest to the goal.[2] One pitfall of the paper is the lack of testing on larger spaces with more shapes that are more complex, as they only reported testing their algorithm on a space which had three pyramidal shapes which are relatively simple when compared to terrain of the real world.

In the real world weather can affect the trajectory of an aircraft and alter the idealized route from a linear solution to something which is non-linear and is dynamic depending upon inclement weather in the vicinity of the flight path [3] One challenge is integrating weather data in order to train a neural network. Solutions to this problem have been investigated and one such solution is using a Big Data framework [3] to perform data fusion of NextGen weather products. This service provides weather data in real-time making it a beneficial addition to simulations of flight trajectories and in use on board real-time systems, constantly feeding information to the agent flying the plane to make rational decisions as fast as possible. This is similar to the way this paper's flight model integrates US weather data and feeds it to the AI. The research done is for a whole flight trajectory, whereas the simulation of our model is real-time

and during the flight in which it follows a predetermined flight plan. Looking at the results from this paper, the Big Data approach of implementing this data in an effective way for computation time is effective for preflight calculations as at the max number of variables included and used in the calculation for the flight trajectory, 11, the computation finished in just under 120 seconds.[5] However, this will prove too long and is why the flight simulation of this paper will use a reduced visual graph[2] which takes into account the closest and most threatening weather hazards.

The time spent in computation needs to be reduced to about 12 seconds, which is how long it takes for the Dynamic Weather Reroutes (DWR)to be updated [3]. The goal in this paper is to show the effectiveness of DWRs and why the FAA should implement them on modern commercial airplanes. By implementing this model in flight simulations, DWR was able to save the plane fuel costs and reduce time when considering a reroute.

In [4], the authors focus on Multi Agent Path Finding (MPAF) systems that do their calculations in real time during the flight as opposed to MPAFs which do their calculations before the flight starts. The overall goal of their simulation was to run as fast as possible with an increasing number of agents in the simulation. The results of the experiment show that when using geometric models the time taken to calculate solutions depending on the number of options stays low, below 50 seconds when options ramp up to 500. Whereas, the voxel model, a method which uses linear algebra and takes a collection of coordinates, increases faster in the time taken for calculations when the number of operations increase, almost 200 when the number of operations reaches 500. This time is improved with pruning for both models, but the geometric model is still faster than the voxel model by 100 seconds when there are 500 operations.[5] Their geometric model is much better in time than using voxel. This will be similar to our approach in the sense that the system will work in real time and will have multiple agents.

Paper [6] introduces a new algorithm which takes two already known algorithms for path finding and combines them into an algorithm called Heuristic Lazy Theta* (HLT*). It works to make the computation time faster by reducing the number of line-of-sight checks present [6].

Paper [7] discusses the changes in flight behavior of planes based on climate, and this affects how flights are routed internationally. These types of changes are difficult to quantify without probing planes going through a route and collecting data manually. This will affect our work, in that our data for turbulence will have to be dynamically rendered relative to nearby weather changes (see [8] for more details, as it covers a very similar set of problems).

Papers [9] and [10] discuss how dynamic routing with imperfect data can be solved within a simulated space, like in video games. Given that our data set is going to be derived from simulated data, these are very important experiences. While these papers discuss well-known and trusted algorithms, they are often in simple, discrete, 2d spaces or along a set navigation mesh, which is only somewhat similar to what we are dealing with in routing airplanes.

Paper [11] discusses whether or not it is possible for real time routing to exist with dynamic weather systems. The idea has been tried but has not yet been fleshed out to its fullest.

Data Set

The data for this airplane routing simulation will be contained in a turbulence map. This will carry the pre-calculated or observed wind directions for training, as well as a height map for observing the influence of the ground structure (wind flowing into a mountain generates an updraft, and certain other trends generate downdrafts etc.) on a simulated plane. This is all encoded onto an image. The turbulence map image will be an RGB image, where the RGB values correspond to X,Y,and Z coordinates. This means that a turbulence map would look similar to a heatmap, but the environment could pull any coordinate from the image to determine the level and direction or turbulence/wind around that point. The heightmap will be a black/white (greyscale) image, where a darker section will mean a higher point on the map. This is mostly important for determining air pressure around a point, since a local downdraft will cause a drop in air pressure assuming the same altitude is held, meaning a slower speed of the plane and therefore a slower route. To further pre-process the data, it will be possible to determine the total air pressure and wind onto a single data structure using these two images, but it would take up far too much space to hold all of that on the program's RAM, so a better solution would be to only calculate the air pressure and wind when it is absolutely necessary as will be described later. So far the environment's input has been discussed, but the plane/player itself will only know the turbulence and air pressure at its specific point, its coordinates on a map containing both the start and end points of its path, and the coordinates of the end point of its route. This map will be generated relative to the position of the plane from its starting point, and this map will contain no data to the player until it is within range of exploration, and it will be forgotten once it falls out of range. This way, the input to the plane is left at a minimum, and the total amount of memory held by the player is very simplistic.

The algorithm to find the fastest or most fuel-efficient path for the model using only this information will be done using a pathfinding algorithm. The model will only 'see' its current position, its destination, and the turbulence/air pressure at its position. It will also be able to calculate its own current velocity and acceleration with this information, so this is already known as well. These last bits of information will be calculated using a starting velocity, along with the work done by turbulence and engine since departing, This will essentially be calculated as a difference equation, meaning velocity[n] = (velocity[n-1] + force[n-1]/(mass_constant))*time_elapsed. Where time_elapsed is the difference in simulated time (in seconds or minutes) between the frames of calculation. This time will be held as a constant, but we plan to have this be affected by initial user input, so as to have a "roughness" of how the end result is calculated vs the time it will take to do so. So a large time roughness would relate to a very imprecise calculation of the result, but it will do so much faster than a small roughness would. But this will be simply for displaying the end results in a manner that is understandable, and to adjust for computing power in an area where that might be limited (for instance, if this software/AI were to be used in a very remote country, or during a dramatic slowdown of processors in an airport). With this in mind, the algorithm can now be described in depth.

Algorithm

To find the current velocity of the plane in an amount of turbulence, a line integral would be very useful. A line integral defines the amount of work done to an object given a vector field of force. This is specified by the integral of F dot-producted with the direction of travel, or (integral) F . (dr). However, this only works if we have a continuous vector field. Instead however, we only have discrete coordinates in both position and force of a vector related to turbulence. There is no real-world solution to this issue, as there will never be a perfect math of the exact speed and direction of wind at every given infinitesimal point in space. Therefore, there will have to be some amount of estimation done at this level. This will be discussed later in this section but the basic formula for work done against the plane in a reference area will be Work =((F_0 - direction_at_origin) - (F_{n-1} - direction_at_end)) * D_{Total}. Where the distance traveled refers to the simulated distance traveled in that reference frame. These reference frames will be determined based on the previously mentioned time constant entered by the user, as well as a suggested holding speed for the plane.

These frames are essentially a mesh of nodes on a graph, relating to coordinates on the turbulence/height maps that the plane itself does not have access to. The plane will use its current observed turbulence and air pressure to place costs on each of the nodes in this mesh, and will then calculate the lowest cost path from this mesh. These costs will be related to this formula. Node_cost = K_1 * (work_done_by turbulence) + K_2 * (change_in_distance_to_destination) + K_3 * (change_in_velocity_at_node) + K_4 * (signed_height_change). This is a tentative formula that will be discussed further at a later section, but these constants K_1 - K_4 will be solved for by a genetic algorithm. The work done by turbulence is useful for determining the amount of fuel used by the engine used to maintain a certain minimum speed. The change in distance to the destination has obvious uses, but overvaluing immediate distance change may result in a loss of efficiency if the path is not a straight line. The change in velocity is solved for separate to work done, since it is more strongly correlated with the actual time difference it takes a plane to go from point A to point B. The signed height change is useful, since a plane will want to fly at a near constant distance from the ground for the majority of its route, so changing too much going too high or too low will negatively impact fuel usage needed to maintain a certain altitude. All of these values are only calculated using the current reading from the plane. Once this new node is decided on by the plane, it will become the new position of the plane, and a certain time will be added onto the total time of the route.

This time is added based on the actual values of the turbulence and air pressures in between the original position and the new one. In the simulation, this will be done by using the “Force_at_end” and “direction_at_end” values as the ones the plane is now reading at the new position, and doing all of the work and new actual velocity calculations using that. This way, the actual turbulence and height map references are pulled only once per reference frame, and the size of the reference frames in relation to physical distance will have to be adjusted in relation to the current computing power available. In the real world, a plane will not need to actually calculate the new velocity and time taken to get to a new point, since it could simply use a sensor to read it’s new velocity. Thus, this inaccuracy caused the discrete nature of the data will not carry over to a real world application, and the calculated vs actual new velocities will be very accurate to one another in most areas. The only exception to this, is where there is a high rate of change in the direction and value of turbulence and air pressure, such that even a tight reference frame will not be able to adjust for it. This is best described by adverse weather patterns, and if

needed, these weather patterns could add a constant cost to those nodes from a map depending on how the genetic algorithm calculates the constants for the rest of the heuristics. Ideally however, high weather areas should be selected against naturally, since the predictions vs the actual speeds of the plane will be very far off, so the AI should not be able to accurately judge how best to make it through one of these areas, and will likely avoid it naturally even without fully knowing exactly where these areas are ahead of time. Depending on which yields the best results in the future, K_5 * (rate_of_change_of_vector_field) or K_5 * (weather_rating) may be added to the node_cost equation.

Training and Testing

This will be a genetic algorithm that solves for the constants that pertain the heuristics in the Node_cost equation of the reference frame nodes. This algorithm will generate many 'players' that have near-random values for those constants. Each of these players will then run this algorithm for the same points as the origin and destination for a route. Once this is completed, these players will be ranked based on the total time it takes to get to the destination, as well as the fuel used to do so (which will be a calculation derived from the air pressure (drag/lift) as well as work done against the plane past it's minimum speed). This could be fine tuned by a user to calculate exactly how to weight these two aspects per company/circumstance (a low income area would weigh fuel usage > time, whereas a high income area might do so differently). The best players of each generation will be 'bred' or selected for by the algorithm. This means that the next generation of players will have constants similar to the best players of the last generation, with some randomized exceptions (or 'mutations'). After a maximum has been reached for a long enough period of time (where time and fuel efficiency have been maximized), the algorithm will stop, and an output will be produced. For testing purposes, will have the algorithm draw on an image the routes taken by the best generation of particles. Highlighting which routes were the most time efficient, which ones were the most energy efficient, and so on. This could also be programmed to view the speed of the plane at various points in the path at RGB values to more easily interpret the information. This could be mapped over the turbulence map as well as the height map to more easily correlate the two (with the turbulence map being displayed similar to how a vector field might be matlab or a similar software, simply for readability). The actual shape of the path would only be useful to an airport or etc.

For this algorithm, mutations should be occurring very often, in order to ensure there is not a quicker path using a different set of values than the local maximum of the current values of the generation. After all of this training is done, the algorithm should also output the actual constants derived for the best players, associated with the values they were best at (best in time efficiency, best in fuel efficiency, etc.) and the result generated by the proposed model will be compared to the time taken through a straight line model. Testing turbulence maps will also be used where the best path is known and obvious, so that we can appropriately determine if the AI is in fact taking the best path as it should, these are cases that would never really happen but they are useful for testing if the model works properly. Examples of these maps would be flat plane with no wind, so that the AI should always make a straight line. Or a map where there is a jet stream in an "L" shape to the destination, and every other path has

heavy wind resistance. A proper resulting player should be able to generate the best path for every map using the same constants it got for the near random practical map. If this is not the case, there is likely a bug in either the methodology or the actual code that must be addressed.

The best way to implement this error checking would be to have an implementation test after each output, that checks a set of the best players against the best routes known for those maps in particular. Only players that solve all of these maps flawlessly could be outputted to an actual airport or practical user, so this should always be checked before its use in any practical setting. Also useful for testing would be the execution time it took to reach that particular AI output, and the amount of processor clocks it would take to get there. Since this software could be used to pre-generate routes using weather data in a localized region, it would be useful to know what kind of hardware would be needed to achieve an accurate result. A very practical application of this software could be to pre-generate routes every morning for an airport, and update their database so that airplanes coming from that port would always follow a near-optimal path. This would rarely need to re-generate an AI, but it could always adjust to real data coming from other planes. Due to this possibility, it may be the case that this algorithm should be re-run for every update in the real turbulence map that can be measured or accounted for, simply using its original measurements as a base. These implementations can be derived in part from papers [3] and [5], where some data can be found as to sensors being implemented in planes that can be sent back to airports to create a database for a live turbulence map, that could be used in our type of algorithm. As such, it is important that this type of live testing would be feasible using our type of AI in a predictable amount of time, and using a known amount of space.

Implementation

The implementation of the proposed model in this paper has a player class which is the agents that are to be trained. Every player has some player data including each player's genes, their current velocity, fuel, direction of travel, position and their destination. The player has distance to destination, an average velocity, a cost mesh reference which tells it how much it costs to move from one point to another. At the start of the program runs, players are initialized and given a destination within the constraints of the height and width of the area specified in the files read in by the input handler. Periodically the player evaluates the area around it and finds the lowest cost path based on whether or not that move puts it closer to the destination and depending on the conditions at that node in terms of turbulence. A balance between these two criteria are what will make this AI change behavior slightly in between the runs, as the weight associated with each variable will change depending on the genes of the players selected from the ranked list of players generated at the end of each run. This ranked list is handled by the output handler which ranks the players performances based on time to get from the origin to the destination.

The input handler on the other hand is working, the main feature of the input handler is its function get vector which when given a coordinate X and Y searches a ppm file for the RGB values associated with the pixel at the given coordinate. These RGB values are used to determine the direction of the wind vector. RGB are assigned to XYZ respectively, since the values of the file are unsigned eight bit integers, we change those to something between -128 and 127, the range of a signed 8 bit integer, depending on its value between 0 and 255. This allows the wind vector to be in any direction. The get vector function also reads the ppm file header and therefore able to retrieve the width and height of the turbulence map, which are the bounds for the area in which the planes are flying. If a plane is asking for a coordinate outside this range, exception checking is put in place to check the asked coordinate with the bounds and detect when the player is trying to reach an out of bounds area.

For the player to understand the cost of moving from one point to another, a cost mesh is constructed around the player at every point so that the player can decide what is a good move. This will also depend on the its genes which will weight what the player favors more. For example if the player has a higher weight on traveling straight toward the destination and a lower weight on avoiding turbulence, then the player will fly straight through the turbulence and go for the destination.

The output handler is used to call the genetic algorithms which breed the players of the current generation. It is also used to print a list of the best players at the end of every five generations to see the progress that the training is making on the model. The routes of every player from those generations are also saved to a text file which can then be interpreted and put into a graph using MATLAB to show a physical graph of how the particles moved during those generations.

For the genetic algorithm, when breeding players, this model takes the particles from the end of the generation and puts them into a group of players to breed. A fitness function which takes the time taken for the particles to reach their destination is used to determine which are fit enough to breed. This function makes it highly likely that those with a low time taken will breed and those with longer times are less likely. Then the selected players will breed and perform crossover, during the crossover there is a small chance for mutation. Any players which would have been selected to not breed would leave a gap in the resulting generations. Therefore, the player roster for the next generation is padded with additional players to conserve the same generation size for each generation.

The data structure used to store these players is a linked list of the players. This includes linked lists for the players, the coordinates and the ranges. The linked list will auto sort each particle as it reaches its destination, that way the linked list are already sorted in fastest to slowest for the output handler when it calls the breeding functions.

The cost mesh that assigns the cost to certain spaces is done. It takes in a function to determine cost and then applies that function across the cost mesh. Using this the minimum cost can be found the player when moving.

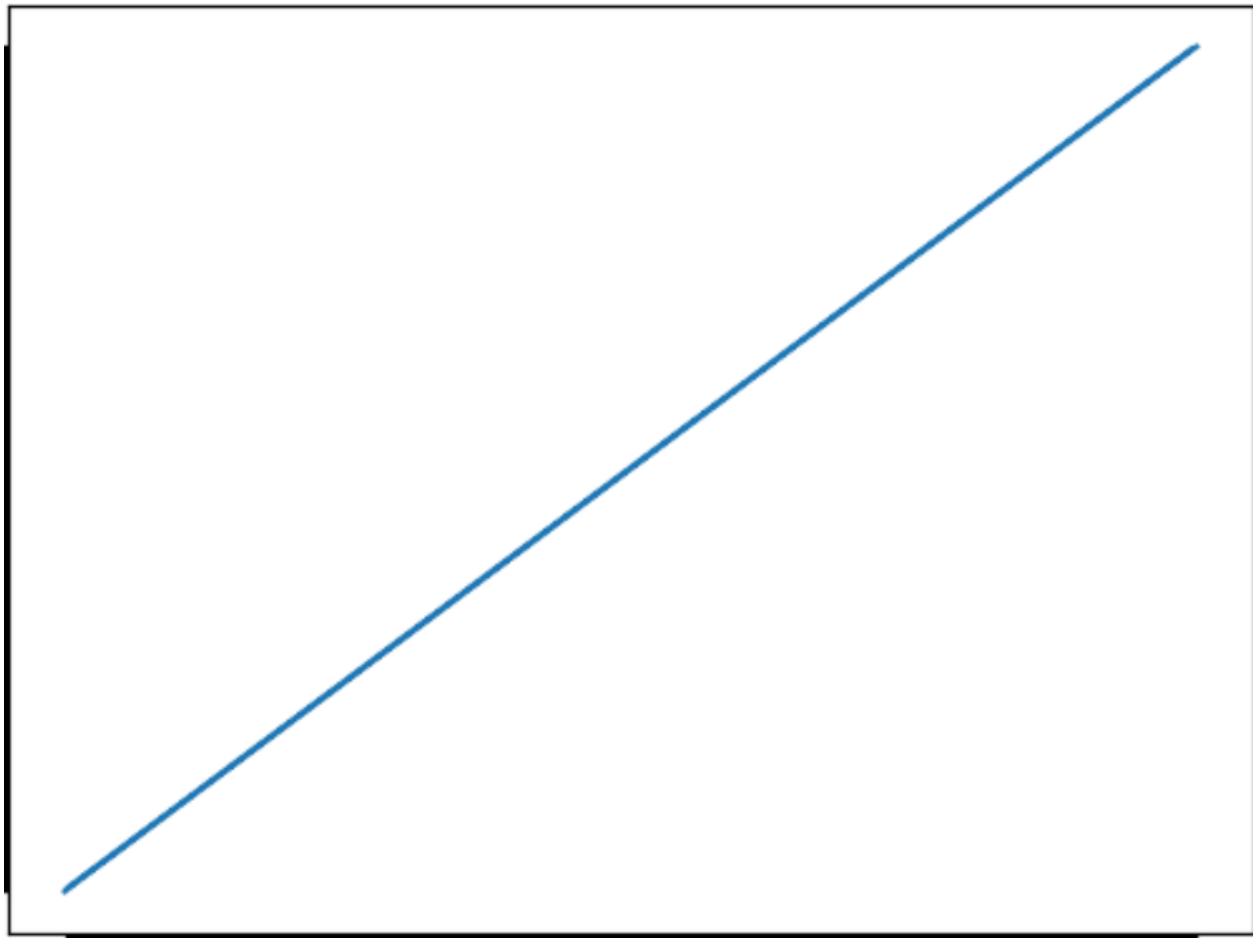
The default physics calculations for work done against the plane in a vector field and other related calculations. These constants are made from statistics for the Boeing 747's mass, average velocity, engine thrust, and other metrics that are going to be used in the simulations for distance traveled by the plane through turbulence.

Results

In each sample test, the average time taken per player was about .83 seconds. This means that a generation of 24 players will take roughly 30 seconds per generation. This test was conducted by timing the program from start to a specified breakpoint after the first generation, and using IDE tools to time this. This time may actually be lower in practicality, since the actual instruments take extra time to execute, and in the terminal the time taken was much lower. This timing from the terminal was not used however, since it is somewhat unpredictable for unknown reasons, whereas the timing from the IDE instruments was much more consistent.

Each particle in every generation has an individually stored route that holds all of its previously visited coordinates in a long linked list. For longer routes, this linked list can be quite large, especially when the particle is too slow in getting to its desired destination. Because of this, each player has a very large size that builds over time. So being especially space efficient was a must, and over the course of this project the average particle size went down roughly 80% from its original size used in the initial testing. The new average particle size hovers around 271KB according to IDE instruments. This is still very large, but not impossible to work with.

The actual output of this program is a long log of the routes taken by each player in each generation. This can be controlled to only output certain generations, or certain players in certain generations. This is very useful for showing exactly the route taken by the best AI in each generation, as well as showing how the AI learns over time. A sample of this output will be shown below for a straight-line test player, or one who only cares about the straight-line distance between it’s position and it’s destination. This graph is generated by a separate python script given the .log file for output.



The origin is in the bottom left and the destination is in the top right. This graph is the path taken by the straight-line test between those two points as a sample.

Also stored is the time taken as extra fuel used by each particle. This is useful since each particle is simulated as a plane flying through a 3D space, and as such the time taken output is fairly accurate to what the actual time taken for the flight should be. Of course, this is impossible to test without an actual plane and known turbulence field to test it against, but it is a useful estimate. The meaning of only holding the extra fuel added instead of the total fuel use is simply that any plane must use some amount of fuel to fight against drag in the air, as well as maintaining lift. Calculating this drag and lift exactly and finding the approximate fuel use per second is not very reasonable without more experimental data. As such, it makes more sense to simply hold the fuel usage as the amount of fuel conserved by going in a tailwind, or spent by being in a headwind, etc.

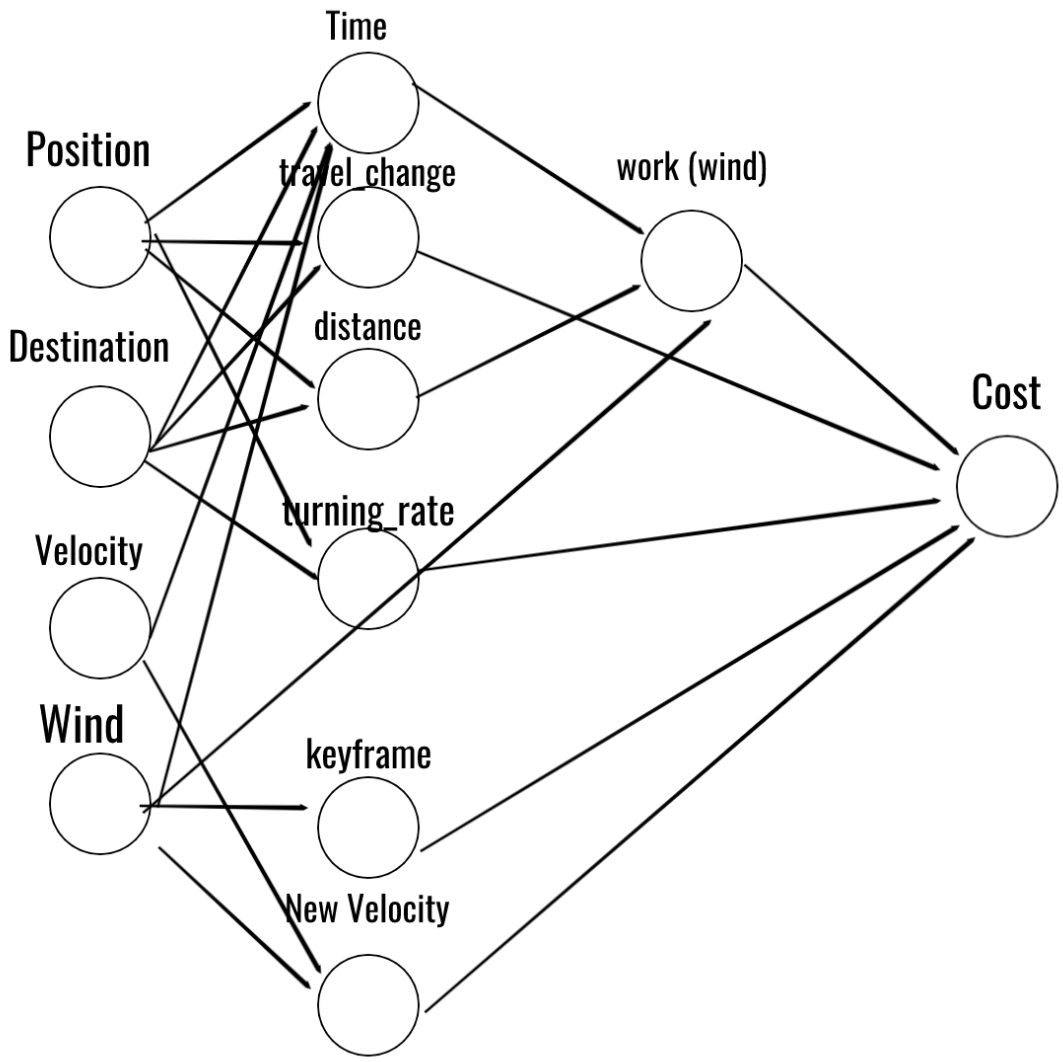
The first generation of particles in this algorithm rarely actually reaches its destination from its origin. On average, only about a sixth of them do, which conveniently is the amount of “genes” carried by the AI, or the amount of hidden nodes in the neural network. After the second generation however, the majority of the particles usually make it to their destination. This is likely because the particles that don’t even make it to their destination do not even breed or affect the next generation, so each generation gains

more players that reach the destination exponentially until all of them do. This makes the execution time faster and more storage efficient after the first few generations.

The largest failure of this AI algorithm is partially in its unpredictability, since it has a highly variable execution time, and partially in its current inability to really work through complex turbulence maps. Because of this, anything with rapidly changing turbulence must be simply marked as “weather” and avoided at high cost. In order to fix this, a more elegant solution for giving the particles information on the rate of change of turbulence must be found, or a different nature of data must be found for predicting turbulence fields (like weather prediction systems). This data exists, but the reliability of this data is unknown in the context of this algorithm, and there is currently no cohesive way to send that information to a particle in a way it understands yet.

Below is the neural network for the inputs to the cost of each position. The weights between the values of the nodes in the hidden layers to the outputs cost is what is being determined by the genes in the

algorithm.



Conclusion

In conclusion this program has been useful in modeling the trajectory of particles in a genetic algorithm as they find their way through a turbulence map to their destination. Even though this program does not have enough information to determine a perfect flight path from two points, it produces a relatively good path given what it believes the turbulence will be at the current moment. This could be used for an algorithm that is constantly updated live, but it is not useful as a predictive measure to find what will be the best flight path using only the current weather conditions. In the future, this functionality could be managed with a weather prediction system as well as this algorithm, but at the moment it would not be useful enough to be used in a real air traffic controller. This is however, a very useful demonstration of how a genetic algorithm can be used in an online problem.

References

[1] K. Button. “A.I in the cockpit.” https://aerospaceamerica.aiaa.org/features/a-i-in-the-cockpit/(accessed)

[2] K. Jiang, L.D. Senevirante, “Finding the 3D shortest path with visibility graph and minimum potential energy” in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Yokohama, Japan, July 1993, pp. 679-684.

[3] Ho, F., Salta, A., Geraldes, R., Goncalves, A., Cavazza, M., & Prendinger, H. (2019, May). Multi-agent path finding for UAV traffic management. In Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (pp. 131-139).

[4] McNally, D., Sheth, K., Gong, C., Love, J., Lee, C. H., Sahlman, S., & Cheng, J. (2012, September). Dynamic weather routes: a weather avoidance system for near-term trajectory-based operations. In *28th International Congress of the Aeronautical Sciences* (pp. 23-28).

[5] R. Madhavrao and A. Moosakhanian, "Integration of Digital Weather and Air Traffic Data for NextGen," 2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC), London, 2018, pp. 1-8, doi: 10.1109/DASC.2018.8569255.

[6] X. Zhang, S. Huang, W. Liang, Z. Cheng, K. K. Tan and T. H. Lee, "HLT*: Real-time and Any-angle Path Planning in 3D Environment," IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society, Lisbon, Portugal, 2019, pp. 5231-5236, doi: 10.1109/IECON.2019.8927202.

[7] V. Grewe1, S. Matthes1, C. Frömming1, S. Brinkop1, P. Jöckel1, K. Gierens1, T. Champougny3, J. Fuglestvedt4, A. Haslerud4, E. Irvine5, and K. Shine5, “IOPscience,” *Environmental Research Letters*, 27-Feb-2017. [Online]. Available: https://iopscience.iop.org/article/10.1088/1748-9326/aa5ba0. [Accessed: 23-Sep-2020].

[8] J. Krozel, A. D. Andre and P. Smith, "Future Air Traffic Management Requirements for Dynamic Weather Avoidance Routing," *2006 ieee/aiaa 25TH Digital Avionics Systems Conference*, Portland, OR, 2006, pp. 1-9, doi: 10.1109/DASC.2006.313782.

[9] A. S. Anisyah, P. H. Rusmin and H. Hindersah, "Route optimization movement of tugboat with A* tactical pathfinding in SPIN 3D simulation," *2015 4th International Conference on Interactive Digital Media (ICIDM)*, Bandung, 2015, pp. 1-5, doi: 10.1109/IDM.2015.7516319.

[10] Cui, X., & Shi, H. (2011). Direction oriented pathfinding in video games. *International Journal of Artificial Intelligence & Applications*, *2*(4), 1.

[11] Grewe, V., Matthes, S., Frömming, C., Brinkop, S., Jöckel, P., Gierens, K., ... & Shine, K. (2017). Feasibility of climate-optimized air traffic routing for trans-Atlantic flights. *Environmental Research Letters*, *12*(3), 034003.