CS-4635: Modeling and Simulation

Final Year Project Report

Global Optimization Approaches to an Aircraft Routing Problem

Group #15

Isaac Mendoza

Jorge Arias

Abraham Diaz

Florian Haule

Department of Computer Science
California State University, Los Angeles

May 15, 2024

Table of Contents

Abstract

Global Optimization Approaches to an Aircraft Routing Problem

M.C. Bartholomew-Biggs1 , S.C. Parkhurst1 and S.P. Wilson1

Numerical Optimisation Centre, University of Hertfordshire, Hatfield AL10 9AB, U.K

In airline industries, one of the most important problems that they face is finding the most optimal route to the destination that is both efficient in time and resources. This problem is called the Aircraft Routing Problem which involves finding an “optimum” flight path between various obstacles from a given origin to a given destination. These obstacles vary from general issues such as geographical features and “no-fly zones”, to even military situations such as avoiding regions around a threat such as enemy radar or missiles. The purpose of this study was to develop an algorithm that aims to minimize a “route cost”, composed of factors such as distance, fuel consumption, and metrics indicating vulnerabilities to risks and proximity to obstacles by comparing a deterministic approach, DIRECT, against two other random searching approaches, TSHJ and ECTS, respectively. After numerous runs and extensive research, it was concluded through numerical data that while all three techniques can be successful, approaching this particular Global Optimization problem in a deterministic way yielded the most reliable results for this specific scenario.

Chapter 1: **Introduction**

The problem that we aim to solve in this project is the aircraft routing problem which in its most simple terms, tries to find the most optimal route between the departure and destination points while considering the various obstacles that could delay the aircraft's arrival. These obstacles can range from expected hurdles such as geological or industrial structures & no-fly zones to unforeseen obstacles like time constraints, dangerous weather conditions & low fuel levels. However, there is also a need to be realistic about what the aircraft can do, so there needs to be constraints for things such as maneuverability limits. These issues are relevant to all kinds of aircraft, whether civilian or military, as they all need a way to get to their destination safely.

There have been some simple solutions to unexpected issues such as avoiding weather conditions via a weather radar but for the most part, these aspects are combined into the “route cost” or “cost value” and used in the planning of the route. To find the optimal version of this route, there have been usages of various global optimization approaches such as direct search techniques, deterministic & random search methods, and tabu search algorithms. These methods aim to find the most reliable and efficient way to calculate flight paths while also taking into account all the various threats & issues. These approaches seek to provide possible routes that usually end up being the optimal routes for different types of missions and geographical areas.

The authors set up the problem using a basic 2D representation of a graph consisting of a number of waypoints and threats. The coordinates of the waypoints were set as the optimization variables under the assumption that the flight path follows straight lines between waypoints. In order to optimize the problem the authors characterized optimality as having the shortest cost as possible. The route was set up by its start and end points and by a number of intermediate waypoints. The authors decided on a direct-search global optimization method due to the fact that the problem produces several local minima and the fact that the objective function is not easily differentiable.

Chapter 2: **Background**

These equations are explained to help readers understand the sub equations and algorithms associated with the Objective function.

$$\phi_j = \cos^{-1}\{\frac{\Delta_j^T \Delta_{j+1}}{||\Delta_j||||\Delta_{j+1}||}\}$$

Turn Angle Equation: This equation is used to find the turn angle between two waypoints. We use this equation in the second part of our objective function in order to check whether the current turn angle exceeds the maximum turn angle. The Δj can be interpreted as l_j known as the jth leg of a waypoint, which is explained more in depth in the next section. It is also important to note that there is an exponent variable T on one of the jth legs. Unfortunately the paper does not expand upon what this T represents and as for our implementation we decided to ignore it and just use the regular jth leg of the waypoint. For the arccos the value is returned in degrees.

Compute l_j as length of leg j. Set $l_{ji} = 0$, $k = 0$, $K_{max} = (\delta\lambda)^{-1}$
If $T_i(u_0) \leq 0$ then $\lambda_b = 0$
For $k = 1, .., K_{max}$
  If $T_i(u_k) \leq 0$ and $T_i(u_{k-1}) > 0$ then
    set $\kappa = T_i(u_k)/(T_i(u_k) - T_i(u_{k-1}))$, $\lambda_b = (k - \kappa)\delta\lambda$
  If $T_i(u_k) > 0$ and $T_i(u_{k-1}) \leq 0$ then
    set $\kappa = T_i(u_k)/(T_i(u_k) - T_i(u_{k-1}))$, $\lambda_e = (k - \kappa)\delta\lambda$
    set $l_{ji} = l_{ji} + (\lambda_e - \lambda_b)l_j$
If $T_i(u_{Kmax}) \leq 0$ then
  set $l_{ji} = l_{ji} + (1 - \lambda_b)l_j$

Algorithm for getting l_{ji}: This algorithm is used to find l_{ji} which is the length that a waypoint crosses into a threat. There are some important variables to note in this equation, the first couple being k and k_{max}. k is simply a step size between the jth leg of a waypoint while k_{max} is the maximum step size possible for the jth leg. This is used to separate the jth leg into smaller parts which makes it easier to check whether any part of the jth leg has crossed a threat boundary and how much of the jth leg crossed it. There is also a function $T_i(u_k)$ that is known as the function of position. This function is not explained in further depth in the paper but our team was able to create a similar function that is most likely what the authors had in mind for this function. We had this function check whether a step size in the jth leg was in a threat point by calculating the distance between the current point in the step size and the current threat's center points. After, we subtracted that distance with the threat's radius giving either a positive or negative value which satisfies the algorithm's conditions for l_{ji}.

$$\delta_j = \sqrt{(\sum_{i=1}^{n} s_{ji}^2)}.$$

Hyperbox equation: This last equation is for DIRECT's hyperbox equation. Hyperboxes are grouped according to a size parameter δj, which is the distance from the centre of the box to any corner. s represents the bounds of the box while n represents the vector displacement for each cycle.

Chapter 3: **Optimization Problem**

$$C = \sum_{j=1}^{n+1} (l_j + \sum_{i=1}^{m} \rho_i l_{ji}^p) + \sum_{j=1}^{n} \nu(\phi_{max} - \phi_j)_-^2 + \sum_{j=1}^{n+1} \mu(l_j - l_{min})_-^2$$

The objective function is used to determine the total cost of the route taken depending on the threats present and the waypoints used. The objective function can be separated into three parts, by each major summation, each determining a specific penalty in the routing problem. We explain each part as follows:

First Part: Checks whether the waypoint passes the threat waypoint and adds a threat penalty to the cost. The first summation starts from 1 to n+1 signifying the summation of the waypoints. The l_j signifies the length between two different waypoints or the jth leg of the route. The second nested summation goes from 1 to m with m being the total number of threats in the mission. Inside the summation is ρ_i which is the penalty parameter associated with the i-th threat. l_{ji} represents the area in which the j-th leg crosses the threat's area. Lastly p is an integer exponent that ensures that C is a continuously differentiable function of the waypoints provided that p is greater than 3. To summarize, this part adds the total route of the waypoints through the summation of the jth legs and adds a penalty value to the current jth leg if that leg crosses any threats.

Second Part: Dictates whether the turn angle exceeds the limiting turn angle before it can increase the cost. If the turn angle doesn't exceed the limiting turn angle then it doesn't affect the cost. The summation goes from 1 to n signifying the total waypoints minus the endpoint. The ν is a penalty parameter associated with turn angles. ϕ_{max} is the limiting turn angle and ϕ_j is the angle between successive stages.The "_" at the end of this part indicates that the expression in brackets is regarded as having the value zero unless it is negative. Finally we square the expression in brackets to make the value positive so it can increase the cost. To summarize this part checks if the turn angle is greater than the limiting turn angle and if it is it multiplies the angle by the penalty parameter and adds the summation to the total cost.

Third Part: Dictates whether the stage length of a waypoint is acceptable before it can increase the cost. If the least acceptable stage length doesn't exceed the length of j then it does not affect the cost. Just like the first part, the summation starts from 1 to n+1 signifying the summation of the waypoints. The μ is a penalty parameter associated with length. Like mentioned in the first part l_j represents the jth leg of a waypoint. l_{min} represents the least acceptable stage length. Like in the second part this part also has "_" subscript setting the expression in brackets to zero if it is not negative. Finally the expression is squared to increase the cost. To summarize this part checks if the length between waypoints is less than the least acceptable stage length and if it is it multiplies the length by the penalty parameter and adds the summation to the total cost.

To summarize the entire objective function, the total cost is equal to the total length of the j^{th} waypoints, the penalty of threats crossed, the penalty of angles bigger than the limiting turn angle, and the penalty of the waypoints less than the least acceptable stage length.

Chapter 4: **Methodology**

The authors set up three different versions of DIRECT (DIRECT, DIRECT-1, and DIRECT-2) each differing slightly but containing all of the changes of the previous. Along with DIRECT, the authors set up two more algorithms, Tabu Search Hooke and Jeeves (THJ) and Enhanced Continuous Tabu Search (ECTS). We explain each method as follows:

DIRECT: The authors decided on using DIRECT because of how its hyperbox function uses Lipschitz constant arguments to decide, at each iteration, which regions of the solution space are worth exploring. DIRECT is set up to use subdivide and identify procedures to get the best route. The subdivide procedure is applied to the best hyperbox worth exploring in which the longest edges of the hyperbox are shrinked. The best hyperbox is chosen through the identify procedure which basically checks if the current hyperbox is less than the minimum or "best" hyperbox. The only user-choice to be made is ε which is set to 10^{-4}. Additionally DIRECT is run with a set number of iterations, I_{max} which is set to 64.

DIRECT-1: Has all of the same configurations as DIRECT but with slightly more changes. This version employs a minimum box-size of 10^{-3}. It also uses an aggressive search if 100n function evaluations have not yielded a significant improvement to f_{min}. Two aggressive searches are allowed if no reduction in f_{min} is observed after 100n evaluations the algorithm terminates.

DIRECT-2: Has all of the configurations of DIRECT-1 but with a few changes. This version performs only one subdivision of each potentially optimal box per iteration (rather than tri-secting parallel to all the longest sides). This makes each iteration less expensive than DIRECT-1. Additionally I_{max} is set 128 instead of 64 like in previous iterations.

THJ: THJ uses a search function which is provided in the equation below to find the global minimum. Each cycle explores random directions away from x^k and if the z for that cycle is the point with the lowest function value given by r global searches then the next cycle centres its explorations on that z. The cycle repeats until all m cycles are complete. The exploration directions are modified with the use of a "tabu list" to guide the direction.

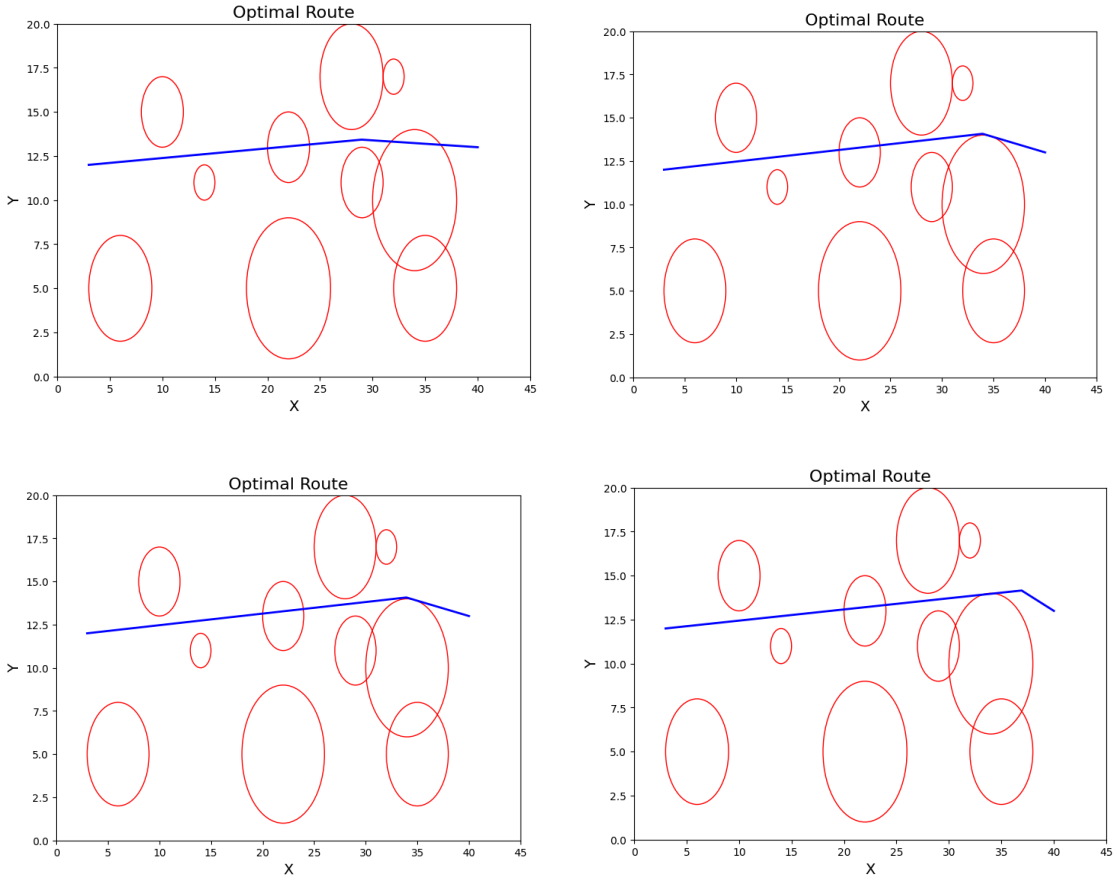
$$x^{(k+1)} = x^{(k)} + \lambda(z_m - x^{(k)})$$

ECTS: ECTS uses a random initial guessed route which then goes through a diversification phase. In this phase the authors used hyperspheres with neighbouring points $x_1 \ldots x_n$ being generated in the "shells" between the hyperspheres. The generated neighbor with the lowest function value becomes the next value to be iterated over. After a specific number of diversification stages have taken place without yielding a new promising point, ECTS uses a intensification stage centered upon the element in the promising list with least function value.

Chapter 5: **Results and Discussion**

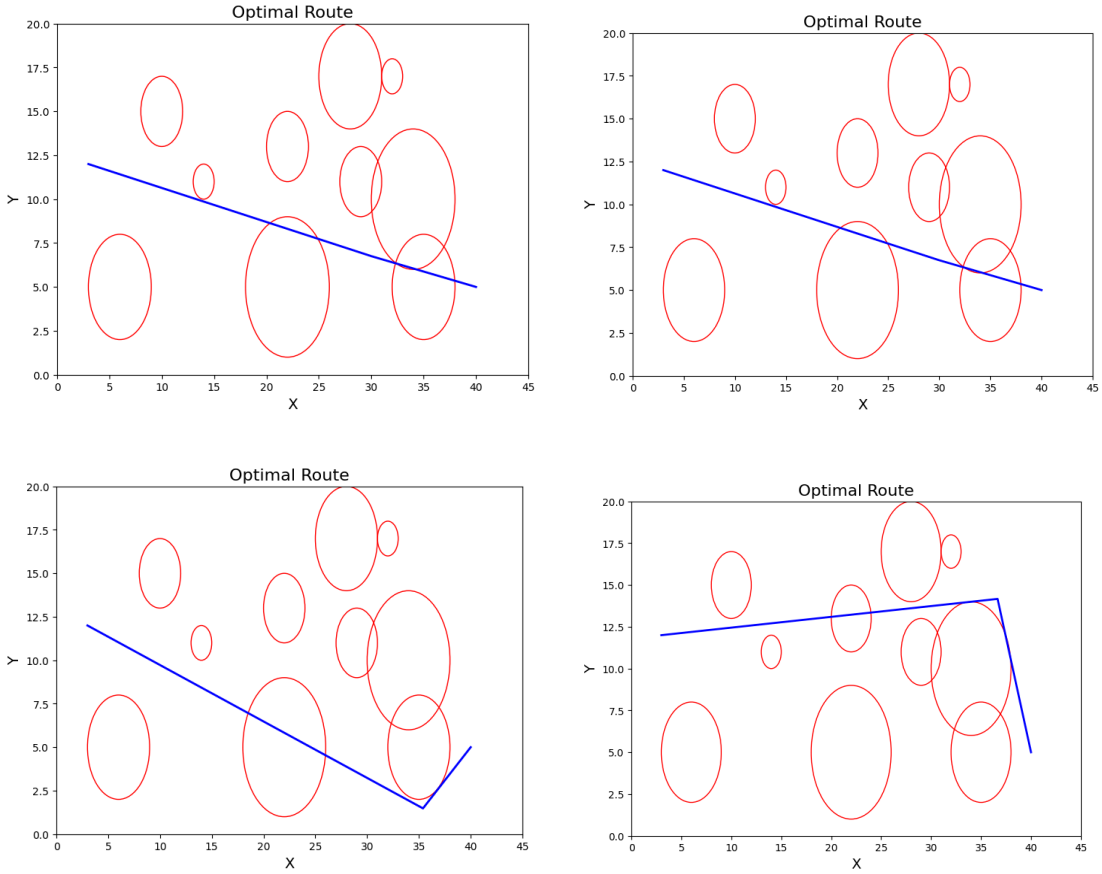
The paper listed eight problems and their solutions that we have tried to replicate. We noticed when trying to implement the problems that none of the problems set conditions affected DIRECT in any way aside from the Mission conditions. This was mostly caused due to DIRECT not having any way to use the set initial guess waypoints, as that feature was only able to be implemented in ECTS and THJ methodologies. Another reason was that since DIRECT's optimization method chose waypoints that had led to the same conclusion despite the different penalty parameters. We assume that this may be because DIRECT chose waypoints to reduce the cost by prioritizing having minimum turns and acceptable stage lengths to reduce the second and third parts of the objective function to zero. So for problems 1-4 which use the first mission parameters the results are very similar and it is the same with problems 5-8 which use the second mission parameters. It is also important to mention that the following results only cover the original DIRECT and not DIRECT-1 or DIRECT-2. We will now go into the results for each mission.

Mission 1 Graphs:



For Mission 1 we have four results, from left to right each graph represents a different ρ value from 0.01 ,0.04, 0.16, and 0.64 respectively. The conditions for mission 1 was a starting point of (3,12) and (40,15). We can see that in the graphs with the low ρ the waypoint tends to go through the last threat but as we increase the ρ value the waypoint tries to avoid the last threat. This is probably due to the increased penalty in the cost value when crossing threat points. Interestingly, each of the waypoints tend to not avoid the middle threat and instead chooses to go straight through it in each iteration. The cause for this might have to do with the turn angle making it more expensive to avoid the threat then to just go straight through it. The cost values from left to right respectively were 37.1610, 37.2191, 37.3839, and 37.8490. These values show that the cost increases with a higher ρ value but relatively stays the same within a 37-38 cost range value due to the adjustments in each waypoint.

Mission 2 Graphs:



For Mission 2 we have four results from left to right each graph represents a different ρ value from 0.01 ,0.04, 0.16, and 0.64 respectively. The conditions for mission 2 was a starting point of (3,12) and (40,5). For the first two ρ values the graphs remained the same with barely any changes between the two graphs. This result is probably because it would have been more expensive to take any other path due to the turn angle and stage length costs. For the third ρ value we see the graph tries to avoid the last threat and turns just to avoid it. Interestingly, the graph doesn't avoid the first threat and goes deeper into the threat than the first two graphs even with a higher penalty. For the last ρ value the graph takes a very similar route to the graphs seen in mission 1. We can see that it crosses the same exact threats and avoids the last threat like in the first mission but does a sharp turn to get to the last coordinate. The cost values from left to right respectively were 37.9074, 38.6723, 41.6789, and 44.1260. These values show that the cost increases with a higher ρ value and increases more than the mission 1 costs with a range from 37-44.

From both mission results we can see multiple patterns. The first is that the routes taken typically do not want to do any turns and if the route needs to turn it prefers to turn only one time in each run. Another pattern in the turns is that most turns are sharp rather than shallow and could be seen as violating the maximum turn angle causing an increased penalty cost to the total cost. The last pattern we were able to find is that the waypoints seem to go into threats that are located in the middle of each graph, probably having to do with turning causing too much of a penalty.

In comparison with our results to the authors, we were able to achieve slightly better results than the authors in some cases. The author's results for mission 1 was 38.1, 37.5, 37.6, and 37.5 going from ρ value from 0.01, 0.04, 0.16, and 0.64 respectively. Our results for the same ρ values were 37.1610, 37.2191, 37.3839, and 37.8490 which shows slightly better results for the first three ρ values. For the last ρ value we were unable to match the 37.5 with our result being slightly worse than the authors. For the second mission the authors only showed one result with ρ value 0.64 which gave a cost of 45.2. Our result gave a cost of 44.1260 giving a slightly better cost value. We

believe the differing cost values occurred due to two reasons, the first being that the DIRECT they were using was from 2003 while we used the most recently updated 2024 version. The second reason might be because of a lack of info for the T variable used in the objective function sub equations, this might have created differing values for the penalty term leading to different results.

Chapter 6: **Your Implementation**

Implementing the optimization algorithm was straightforward but challenging due to the multiple equations associated with the actual objective function. We first studied the variables and parameters associated with the objective function. Some of the variables were obtained through other equations listed in the paper. Once we had a basic understanding on how each part of the objective function worked, we started creating the pseudocode for the objective function splitting each part into separate functions. We explain the challenges with each part of the code as follows:

Part 1: For part 1 one of the challenges we came across was obtaining the value for variable l_{ji}. The main issue that stopped us from obtaining this value was trying to figure out how to check if l_j was inside a threat and how close to the center the waypoint was. To solve this we created a function to create a variable "distance_between_center" that has the value of the distance between a point and the center points of a circle. We then subtracted the radius from the "distance_between_center" which returns a negative value if the point is in the circle and a positive if the point is outside of the circle. This allowed us to implement the equation for getting l_{ji} into our code since it checks whether the point is in a threat and assigns l_{ji} a certain value based on each outcome.

Functions for getting distance between center:

```
def distance_between_points(point1, point2):

       if isinstance(point1, (tuple, list)) and isinstance(point2,
(tuple, list)):

        x1, y1 = point1

        x2, y2 = point2

    else:

        x1, y1 = point1, point1

        x2, y2 = point2, point2

    dx, dy = x2 - x1, y2 - y1

    distance = math.sqrt(dx**2 + dy**2)

    return distance

def check_inside_threat(x, y, cx, cy, r):

    point = [x, y]

    center = [cx, cy]

    distance_from_center = distance_between_points(point, center)
```

```
    return distance_from_center - r
```

Part 2: For part 2 we came across a problem with obtaining ϕ_j ,more specifically we couldn't figure out what the value for l^T_j. We weren't sure what the authors used for T, the only mention of T is used to find the "distance between center" variable. When we tried to implement that method for T we found that for values not in the radius would sometimes be greater than 1 causing a math domain error for the equation. In the end, we decided on using l_j for the value of l^T_j but we are not sure if there is an actual meaning to l^T_j with a different value.

Part 2 implementation:

```
        a = 1

        for a in range(len(waypoints) - 1):

          newPoint1 = waypoints[a]

          newPoint2 = waypoints[a+1]

          ljNew = distance_between_points(newPoint1,newPoint2)

          angleJ = math.acos((lj * ljNew)/(abs(lj)* abs(ljNew)))

          angles = (angleMax - angleJ)

          if angles > 0:

            angles = 0

          reduce_angle += penalty_term_v*(angles**2)
```

Part 3: For part 3 we didn't come across any problems and were able to implement this part with no errors.

DIRECT: While using DIRECT, a problem we had was the differing values from the authors' results compared to ours. While we were never able to figure out the reason why our results differed, we theorize that the difference in DIRECT versions may be the reason why our DIRECT gave different results. The paper was published in 2003 while we began this project in 2024, the versions in DIRECT may have updated and the hyperbox function may have updated to be more accurate in the following years. Of course, this is all up to conjecture but we feel like if we were either given the DIRECT algorithm in the paper or found the pseudo-code of the paper's optimization we would've been able to find the issue that caused the different values.

You may find the link to our code here on Google Collab:

https://colab.research.google.com/drive/1ekroivH2huT9MSsdYUKsL4-BG7nypAbt?usp=sharing

Bibliography

[1] Bartholomew-Biggs, M. C., et al. “Global Optimization Approaches to an Aircraft Routing Problem.” European Journal of Operational Research, vol. 146, no. 2, 2003, pp. 417–31. ideas.repec.org, https://ideas.repec.org//a/eee/ejores/v146y2003i2p417-431.html.