HIGH SCHOOL EDITION
Journal of Student Research
Volume 12 Issue 1 (2023)

Finding an Optimized Flight Path for an UAV to Seed a Fire Affected Area

Abhishek Kaushikkar[1] and Joshua Whitman[#]

[1]Leland High School, USA
[#]Advisor

ABSTRACT

Fires have devastated and cleared many areas of vegetation, and much of the terrain impacted is inaccessible by foot. As a result it is difficult and inefficient to re-seed these areas from the ground. One way of getting around this problem is by using an Unmanned Aerial Vehicle(UAV). Aerial vehicles are not hindered by topographical and organic features. However UAVs cannot remain in flight for extended periods of time. To maximize the UAVs potential, steps must be taken to limit its flight time. Our paper aims to find an algorithm that computes the most energy efficient path for the UAV to follow in order to re-seed the fire cleared areas in a forest and we calculate based on a few constraints. These constraints being that the UAV can only seed a circular area with a fixed radius, the UAV can translate in straight lines, the area is an N×N square, and the terrain varies in altitude, and the UAV translates at a constant velocity. We first calculate an array of nodes that the UAV can seed over, to get maximum coverage of the un-vegetated areas in the field. We then test three different algorithms to find the path that consumes the least amount of energy, based on our energy consumption model. We then compare the algorithms based on the energy consumption of their calculated paths, and their computation time.

Introduction

Fires have devastated and cleared many areas of vegetation, and much of the terrein impacted is inaccessible by foot. As a result it is difficult and inefficient to re-seed these areas from the ground. However aerial vehicles can do this task without facing as much of an obstacle from topographical features. UAVs are perfect for this.

a. Unmanned aerial vehicles or UAVs, are able to linearly translate in any direction, and have the ability to perform actions such as hovering in place, and flying straight up. This makes them desirable for our purposes. UAVs do face an important constraint, they have an onboard power system, and they require a lot of energy. Without the ability to recharge in flight, they become limited and thus they cannot remain in flight for extended periods of time. To maximize the UAVs potential, steps must be taken to limit its flight time.
b. In this paper we explore the following question. When given a mapped region, what solution computes the most energy efficient path for a UAV to seed a semi cleared area. We introduce a few constraints:
   1. The UAV can only seed a circular area with a fixed radius.
   2. The UAV can translate in straight lines.
   3. The area is an N×N square, and the terrain varies in altitude.
   4. The UAV translates at a constant velocity.
   5. The difference between the UAV's altitude and the altitude of the terrain is constant.
c. Background/literature Review: The algorithm utilized will have to be a Coverage path planning algorithm or CPP, based on a Traveling Salesman Problem or TSP. Quite a lot of research has been done on coverage path planning algorithms already. For example: *A Survey on Coverage Path Planning for*

Robotics, discusses the use of heuristic algorithms that guarantee complete coverage of the free space. The paper discusses online algorithms that break down targetspace into subspaces called cells to iteratively construct a topographical map of the environment.[1] Another paper; *Path Planning for UAVs*, uses UAVs to traverse from point A to point B without crossing any obstacle points, in the shortest distance. The algorithm proposed creates a graph of suboptimal paths and then weights are assigned to each edge based on the edge length and probability of detection from obstacle points. Using these weights the algorithm finds the optimal path from the start point to the end point.[2] There are also many existing TSP algorithms, the first utilizing a dynamic programming algorithm. The paper *Survey of Methods of Solving TSP along with its Implementation using Dynamic Programming Approach* Goes over the application of many different exact and heuristic algorithms, and their implementation towards the Traveling Salesman problem.[3] The Traveling Salesman problem is a np hard problem meaning that there may not be a way to compute an exact solution every single time. The Dynamic Programming approach is an exact solver algorithm. The other class of algorithms are heuristic algorithms, which only promise a feasible solution, but this solution is not guaranteed to be the most optimal. The two heuristic algorithms discussed in this paper are the simulated annealing algorithms and the Nearest Neighbors algorithm. Literature also exists on route based UAV energy consumption. *Comprehensive Energy Consumption Model for Unmanned Aerial Vehicles, Based on Empirical Studies of Battery Performance* Discusses energy consumption in multi rotor drones in the takeoff, translational, and climbing phases of flight. [4]

Methodology

I. Field Generation: An accurate 2D field with both vegetated and unvegetated areas is needed to perform analysis on the UAV's pathfinding. In this case, the dimensions, (N,M) are constant and the field takes the shape of a square so N=M. For our purposes the field was set to 100 pixels by 100 pixels. Next, the number of unvegetated patches and their maximum radius needs to be found. The number of patches is set to 12 and the maximum radius is randomly found using a normal distribution which is produced by the normal density function:

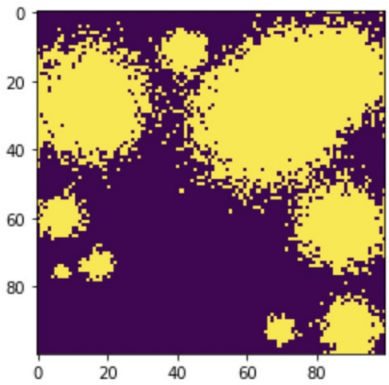
$$p(x) = \frac{e^{\frac{-(x-\mu)^2)}{2\sigma^2}}}{\sigma\sqrt{2\pi}}$$

where μ is the mean, and σ is the standard deviation. Because N and M are 100 $\mu = 5$ and $\sigma = \frac{\mu}{2}$. The next step is to the coordinates of each patch which x is simply a uniform random integer in the interval [0,N] and y is a uniform random integer in the interval [0,M]. The size of the patch is

patch_size = $patchradius^2 \times 100$

From there on, each individual pixel x in the patch is a random normal integer where $\mu = patchx$ and $\sigma = patch\ radius$. Each individual pixel y in the patch is a random normal integer where $\mu = patchy$ and $\sigma = patch\ radius$. Each generated pixel has to be a new pixel, as well its x and y being in the interval [0,100]

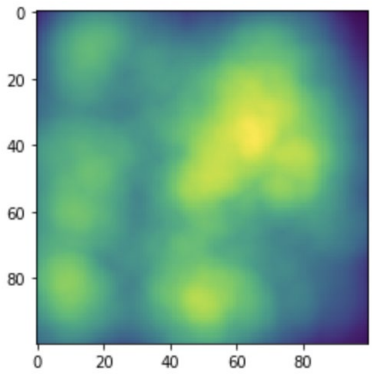
Example of a generated field where yellow is unvegetated, and purple is vegetated:



The second part of field generation is creating a terrain map. Many methods were considered, however the method the yield the most promising result was the hill algorithm which picks a uniform random point on a flat surface and raises a number of hemispherical hills with a random uniform radius and random uniform height that is equivalent to the radius[7]

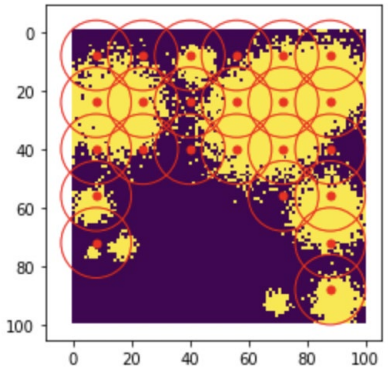
As the number of hills increases, the terrein begins to become more realistic and random. The number of hills chosen in this paper is 200.

Example terrain map with 200 hills where blue indicates lower altitude and yellow indicates higher altitude:



II. Choosing nodes: The next step is to choose the placement of nodes that maximizes the UAV's coverage of the un-vegetated areas. The first assumption is that the area that the UAV can seed in one seeding

spray is a circle with a constant radius r. There are two geometries in which the nodes may be placed to maximize coverage. the first is where each node is equidistant from one another as well as being parallel to one another such as shown in the figure below:



The distance between each node d in this geometry is equal to $r\sqrt{2}$

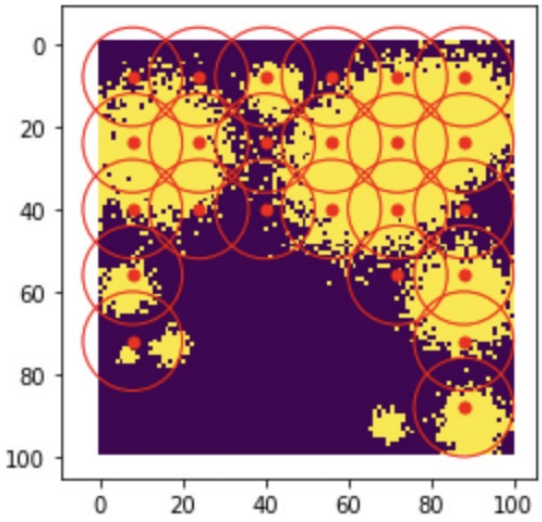
The starting node is placed at the coordinate

$(0+\frac{d}{2}, 0+\frac{d}{2})$

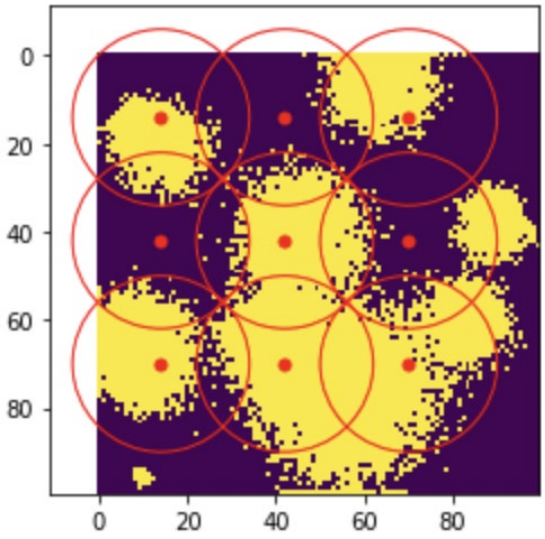
A point is selected as a node on the flight path if the circle around the point, with the radius being the radius of the seeding spray is a circle, contains enough unvegetated pixels to be greater than or equal to the area of the circle divided by 3.86. To reduce runtime, if a point is in an unvegetated area, it is automatically selected to be a node.

$$nodes = \{(x_i, y_j) : x_i = x_0 + r\sqrt{2}i, y_i = y_0 + r\sqrt{2}j, 0 \leq i \leq i_{max}, 0 \leq j \leq j_{max}\}$$

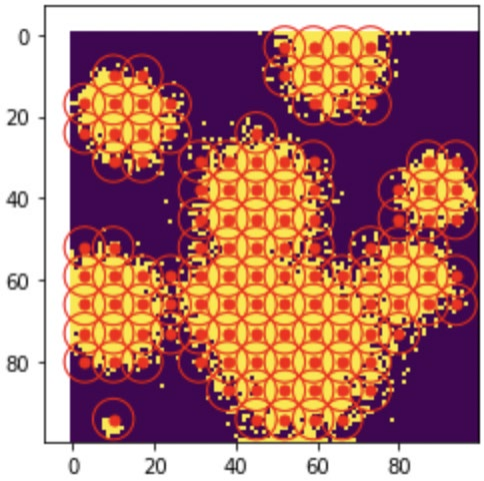
Coverage when $r = 12$:



Coverage when $r = 20$:



Coverage when r = 5:



As shown in the figures above, as the r decreases, the accuracy of coverage and the percent of the covered area increases.

III. Computing energy matrix of all nodes. An energy matrix is based on the concept of a distance matrix which is a nonnegative, square, symmetric matrix with elements corresponding to estimates of some pairwise distance between the sequences in a set. [5]

We calculate the distances between each node, and then create a matrix where the value at Matrix[x,y] equals the distance between points x and y. In an energy matrix, we calculate the energy consumed in a flight between each node instead of distance. More on the Energy calculation in section V.

IV. Traveling Salesman algorithms. The first algorithm we implement is the dynamic programming algorithm which provides an exact solution to the traveling salesman problem. For this problem, each node is a point that needs to be visited by the algorithm. In the dynamic algorithm for TSP, the number of possible subsets can be at most $N \times 2^N$. Each subset can be solved in $O(N)$ times. Therefore, the time complexity of this algorithm would be $O(N^2 \times 2^N)$. We implement this algorithm using the library python-tsp.

The second algorithm we are implementing is the simulated annealing heuristic algorithm. This algorithm is a heuristic algorithm, which is a kind of algorithm that sacrifices optimality, accuracy, precision, or completeness for speed. This approach is far more effective on sets with a large number of values. The simulated annealing algorithm works by

1. Create the initial list of cities by shuffling the input list (ie: make the order of visit random)

2. At every iteration, two cities are swapped in the list. The cost value is the distance traveled by the salesman for the whole tour.

3. If the new distance, computed after the change, is shorter than the current distance, it is kept.

4. If the new distance is longer than the current one, it is kept with a certain probability.

5. We update the temperature at every iteration by slowly cooling down.[6]

The final algorithm implemented was a variation of the Nearest Neighbors algorithm. Like the Simulated Annealing algorithm, the nearest neighbor algorithm is a heuristic algorithm. This means that it is much faster and space efficient than an exact solution, however it is less accurate. The algorithm implemented is recursive.

These are the steps of the algorithm:

1. Initialize all vertices as unvisited.
2. Select an arbitrary vertex, set it as the current vertex u. Mark u as visited.
3. Find out the shortest edge connecting the current vertex u and an unvisited vertex v.
4. Set v as the current vertex u. Mark v as visited.
5. If all the vertices in the domain are visited, then terminate. Else, go to step 3.

The time complexity of the nearest neighbors algorithm is $O(N^2)$

V. Energy cost. The final step is to calculate the Energy cost of the solution for the UAV.[4] Total energy is represented by:

$$E(t) = \int_0^t Pdt$$

Power consumption when the UAV is hovering is represented by:

$$P = \sqrt{\frac{(mg)^3}{2\pi r^2 \rho}}$$

where P is power, mg is the Force of gravity acting on the UAV. Since the UAV is hovering in place,

$$Thrust_{UAV} = mg$$

When the UAV is translating horizontally and climbing:

$$Thrust_{UAV} = \frac{mg + \rho dA|v|^2}{cos(\phi)}$$

When the UAV is translating horizontally and descending:

$$Thrust_{UAV} = \frac{mg - \rho dA|v|^2}{cos(\phi)}$$

where ϕ is the roll angle of the UAV, d is the drag coefficient of the UAV, and v is the climbing velocity of the UAV. The climbing velocity is based on the proportion of climbing to horizontal displacement that the UAV needs to do between point a and point b.

$$v_v = v_c \ cos(\theta)$$

where v_c is the net velocity of the drone, and

$$\theta = tan^{-1}(\frac{vertical\ displacement}{horizontal\ displacement})$$

ϕ is related to v_h or the horizontal velocity of the UAV, in a way that

$$\phi = tan^{-1}(\frac{\rho dA|v_h|^2}{mg})$$

Therefore, the energy consumed by the UAV between point a and point b is represented by:

$$E(t) = \int_0^t \sqrt{\frac{(Thrust_{UAV})^3}{2\pi r^2 \rho}}\, dt$$

We can find t by dividing the total displacement of the UAV between point a and point bby the constant $velocity$ of the UAV. In order to find an accurate representation of the energy consumed in a flight over

the terrain between two points, we divided the path line between the two points into equal parts. We then calculated the sum of the energy consumed in all those parts. The total parts can be found by:

$$totalparts = \frac{Distance(a,b)}{pixels}$$

The points on the line is a set where each point is a set number of pixels away from the previous point on the line where point a is the first point and point b is the last point. In our case the number of pixels is four. The times of flight for each section can be represented by the following set:

$$times = \{t;\ t = \frac{D(a,b)}{v_c} \Rightarrow a = points_k \ \wedge\ b = points_{k+1};\ k = Z \Rightarrow 0 \leq k \leq |points|\}$$

The total energy between two points can be represented as

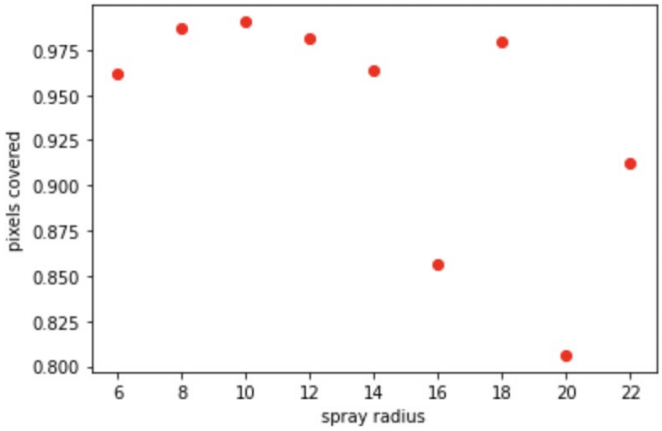
$$E = \sum_{k=0}^{|times|} E(times_k)$$

To evaluate the performance of each algorithm, we will compare the mean energy consumption, and execution time of each solution.

Results

Spray Radius and Coverage

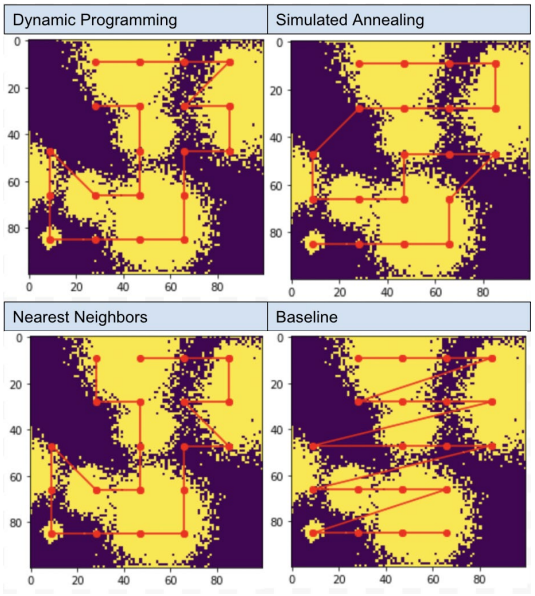
The graph below shows the coverage on the unvegetated area based on the spray radius:



We decided to select 14 as the spray radius for the comparisons of the algorithms as using spray circles with that radius provides coverage similar to smaller circles, while also using less nodes for coverage which is more efficient for the flight of the UAV and for the computations for the path planning algorithms. In addition, this spray radius leads to a range of 12-25 nodes, depending on how cleared the field area is. Although a spray radius of 10 and 12 provides better coverage on average, using those radius's yields a number of nodes that is too large for the dynamic programming algorithm to work with, so a spray radius of 14 was chosen.

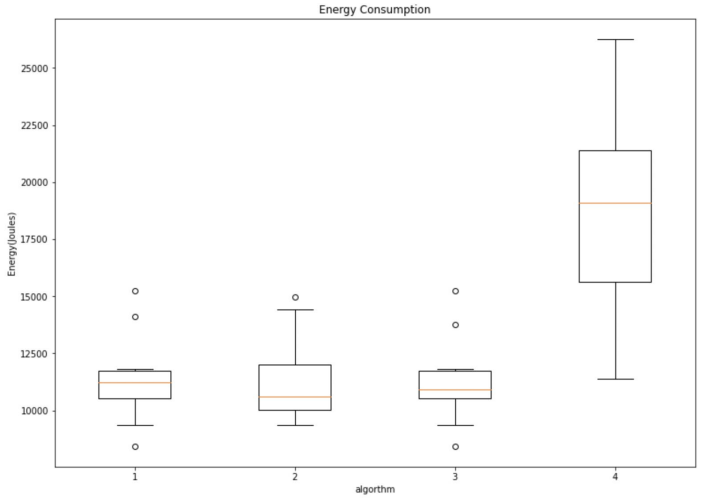
Paths

Each path generated by each algorithm was mapped as such. Here is a diagram that shows the different paths each algorithm came up with, when solving on the same field, where the radius of the circle that the UAV can seed is 14:

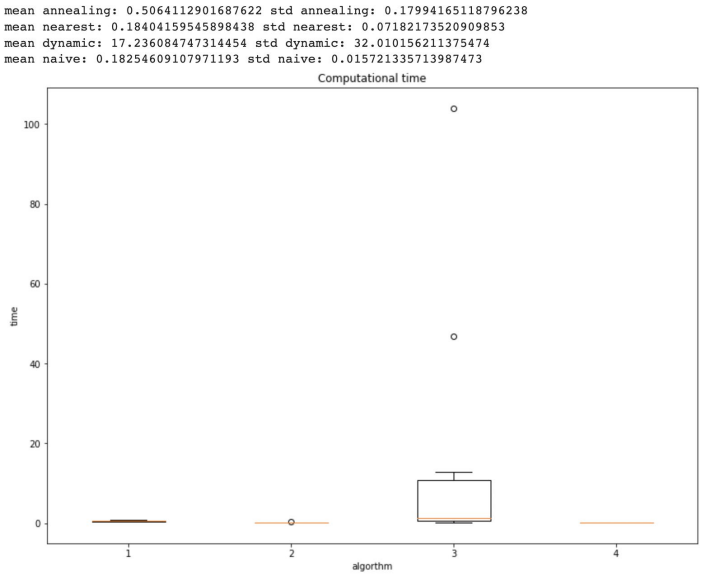


We tested the algorithms on a sample size of 20 different fields. The first variable we measured was energy consumption. As seen in the graph below, the Algorithm with the lowest mean was the nearest neighbors algorithm. In all the plots below, algorithm 1 is the simulated annealing algorithm, 2 is the nearest neighbors algorithm, 3 is the dynamic programming algorithm, and 4 is the naive zigzag approach.

```
mean annealing: 11414.933119749288 std annealing: 1918.2547597032217
mean nearest: 11290.966807097018 std nearest: 1892.2722662868757
mean dynamic: 11310.634513962214 std dynamic: 1873.7512644927924
mean naive: 18595.728785461848 std naive: 4170.652837954479
```



The next variable we measured was the computation time for each algorithm. Although the mean computation time for the naive solution was the lowest, the second lowest was the computation time for the nearest neighbors algorithm.



Discussion

We measured the performance of three different algorithms, and compared them to a baseline of a naive zigzag approach. Based on our results we found that the algorithm that on average consumes the least amount of Energy was the Nearest Neighbors algorithm, with a mean energy consumption of 11290.966807097018 Joules. The dynamic programming algorithm followed just behind with a mean energy consumption of 11310.634513962214 Joules. The simulated annealing algorithm, was almost as efficient as the dynamic programming algorithm as its mean energy consumption was 11414.933119749288 Joules. All tested algorithms soundly outperformed the naive solution baseline, which had a mean energy consumption of 18595.728785461848 Joules. Thus for calculating the most energy efficient flight path, the data suggests that the nearest neighbors algorithm is the best bet, however the dynamic programming and simulated annealing algorithms have very similar performance metrics. With a standard deviation of 1873.7512644927924 Joules, the dynamic programming algorithm provides the least amount of variance in the energy usage for each field it solves. Unsurprisingly the naive solution had the most variance, with a standard deviation of 4170.652837954479 Joules.

In terms of computational time, the fastest algorithm was the naive solution with a mean computation time of 0.18254609107971193 seconds. The nearest neighbors algorithm was almost as quick with a mean computation time of 0.18404159545898438 seconds. When solving problems with larger amounts of nodes, the algorithm had very little variance in its computation time, with a standard deviation of 0.07182173520909853 seconds. The dynamic programming algorithm was the slowest with a mean computation time of 17.236084747314454 seconds. With a time, complexity of O(N2+2N) the computation time of the dynamic programming algorithm increased exponentially as the number of nodes increased. This fact materializes in the standard deviation of the computational times of the dynamic programming algorithm as the standard deviation was 32.010156211375474 seconds. The naive solution was the fastest solution, and its standard deviation was 0.015721335713987473 seconds, the lowest variance out of any of the tested algorithms. Finally, the simulated annealing algorithm was the third fasted, with a mean computation time of 0.5064112901687622 seconds, and it had the second highest variance in computation time, with a standard deviation of 0.17994165118796238 seconds.

Thus, the results suggest that if the UAV can spray a circular area with a radius of 14, the best algorithm to calculate its flight path would be the nearest neighbors algorithm. This is because the algorithm has the lowest

computational time, and on average it requires the lowest average energy consumption. This fact is important as the UAV has a fixed, limited amount of onboard energy. If one were to implement a solution to the UAV seeding problem, it would make the most sense to utilize the nearest neighbors algorithm. However the simulated annealing algorithm is not that far off in the measured metrics so one cannot go wrong with either the simulated annealing algorithm or the nearest neighbors algorithm. Both Heuristic algorithms are great for solving this problem, and are better than the exact algorithm, the dynamic programming algorithm, which is simply too time costly to implement on a realistic scale, especially when the UAV needs to visit more than 20 nodes.

Conclusion

In conclusion, when the UAV can only seed a circular area with a fixed radius, translate in straight lines and travels at a constant velocity, the most effective solution to seeding a forested N*N area with wildfire cleared areas. is to incorporate the nearest neighbors heuristic algorithm. This applies when there are between 12 and 25 total nodes. The simulated annealing algorithm can also be incorporated but its mean energy consumption and computation time was slightly lower than the nearest neighbors algorithm.

There are many improvements that can be made to our experimentation in the future. The first is to do more experimentation on the effects of different spray radius values on the total seeding coverage and the flight path. Currently we have data of the effects of different spray radius's coverage on one field only, an improvement would be to collect more comprehensive data on the spray radius' effect on multiple different fields. Another improvement is to consider different techniques for selecting nodes along the UAVs path. Currently we are only using a grid geometry for potential nodes, however in the future, we could test the effects of staggering each row of nodes. In our experiments, the UAV was translating in straight lines only, the possibility of non linear paths could have an impact on the results of the tests. Finally we could experiment with the size of the field to see how that impacts the node selection and route calculation for the UAV as all our tests were done on a square 100m*100m field.

Acknowledgment

Thank you for the guidance of Joshua Whitman, from The University of Illinois, Urbana Champaign in the development of this research paper.

References

[1] Enric Galceran, Marc Carreras, "A survey on coverage path planning for robotics, Robotics and Autonomous Systems," Volume 61, Issue 12,2013,Pages 1258-1276,ISSN 0921-8890

[2] S. A. Bortoff, "Path planning for UAVs," Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No.00CH36334), 2000, pp. 364-368 vol.1, doi: 10.1109/ACC.2000.878915.

[3] Chauhan, Chetan, Ravindra Gupta, and Kshitij Pathak. "Survey of methods of solving tsp along with its implementation using dynamic programming approach." International journal of computer applications 52, no. 4 (2012).

[4] H. V. Abeywickrama, B. A. Jayawickrama, Y. He and E. Dutkiewicz, "Comprehensive Energy Consumption Model for Unmanned Aerial Vehicles, Based on Empirical Studies of Battery Performance," in IEEE Access, vol. 6, pp. 58383-58394, 2018, doi: 10.1109/ACCESS.2018.2875040.

[5] "Distance Matrix." Distance Matrix - an overview | ScienceDirect Topics. Accessed September 17, 2022. https://www.sciencedirect.com/topics/mathematics/distance-matrix#:~:text=The%20distance%20matrix%20between%20the,N%7D.

[6] Prateek, Peter, Ss, Emmanuel Goossaert, Hakim, Alim, and Zahra Nekudari. “Simulated Annealing Applied to the Traveling Salesman Problem: Code Capsule.” Code Capsule | A blog by Emmanuel Goossaert, July 15, 2014. https://codecapsule.com/2010/04/06/simulated-annealing-traveling-salesman/#:~:text=Simulated%20annealing%20is%20an%20optimization,designed%20specifically%20for%20this%20problem.

[7] .Terrain generation tutorial: Hill algorithm. (n.d.). Retrieved October 1, 2022, from http://www.stuffwithstuff.com/robot-frog/3d/hills/hill.html