# Fuzzy Clustering Based Genetic Algorithm for the Multi-Depot Polygon Visiting Dubins Multiple Traveling Salesman Problem

**Conference Paper** · June 2012

DOI: 10.2514/6.2012-2562



**2 authors:**

Nicholas Ernest
Psibernetix Inc.
**14** PUBLICATIONS **45** CITATIONS
SEE PROFILE

Kelly Cohen
University of Cincinnati
**207** PUBLICATIONS **994** CITATIONS
SEE PROFILE

Infotech@Aerospace 2012
19 - 21 June 2012, Garden Grove, California

AIAA 2012-2562

# Fuzzy Clustering based Genetic Algorithm for the Multi-Depot Polygon Visiting Dubins Multiple Traveling Salesman Problem

Nicholas Ernest[1], Kelly Cohen[2]
*School of Aerospace Systems, University of Cincinnati, Cincinnati, OH,45221*

**A genetic algorithm (GA) and Fuzzy Logic System (FLS) based approach to the path representation of a variant of the Traveling Salesman Problem (TSP), the Multi-Depot Polygon Visiting Dubins Multiple Traveling Salesman Problem (MDPVDMTSP) is presented. Utilizing a hybridization of control techniques, this work effectively and efficiently approximates path planning and visbility problems encountered by a UAV swarm in the constant altitude, constant velocity, two-dimensional case. Benchmarking capabilities only exist for a 20 polygon PVDTSP case (1 UAV), but for this case a 9.8% increase in accuracy along with an order of magnitude decrease in run-time was found compared to the alternative method, despite utilizing a slower computer and programming language. Over 100 runs, the best solution was found 99% of the times, with a 1% chance of settling to a local minima. Comparison opportunities for MDPVDMTSP's are currently not present, but the algorithms work similarly well for this scenario. MDPVDMTSP's with 250 polygons, 20 UAV's, and 4 different depots can be accurately approximated in under 30 seconds on the same machine. Being a combination of approximate methods, a decrease in run-time was expected, however the accuracy of this work shows great future promise for more complex variants of this problem.**



## I. Introduction

UAV's are being utilized increasingly more often as the technology develops. The capability of complete autonomy will push the envelope of UAV (and other robotic) applications even further. In every possible application of UAV's, the elimination of a trained pilot (or in the case of a UAV swarm, a large multitude of trained pilots) brings much greater accessibility, affordability, and portability. In the U.S. Army alone, UAV's of all types have over a million hours of flight time [1,2]. Especially with this increasing popularity, efficient allocation of these UAV's is vital to cost-effectiveness. Working in conjunction with other softwares, the end goal of this research could eliminate the need for a human operator, or at least reduce it to simply monitoring progress and verifying results.

This work focuses on a variant of the Traveling Salesman Problem (TSP), the Multi-Depot Polygon-Visiting Dubins Traveling Salesman Problem (MDPVDMTSP) is the subject of this paper. This extension of Dr. Obenmeyer's Polygon Visiting Dubins Traveling Salesman Problem (PVDTSP) [3] more closely resembles the problems encountered by UAV swarms. Previous publications [5,6] describe the creation of "UNCLE SCROOGE" (UNburdening through CLustering Effectively and Self-CROssover GEnetic algorithm), a genetic algorithm that can solve the TSP effectively partnered with a fuzzy logic clustering system for Multiple TSP's (MTSP). These same techniques are utilized here through a dynamic programming based approach that breaks down the MDPVDMTSP into a set of simpler problems.

### A. Background

If given an effective radius of a UAV's equipped camera, weapon payload, etc., a hemisphere can be created above-ground around a land-based target. Assuming 360 degree capabilities, the UAV entering this hemisphere at any point translates to a successful overpass of that target, in which it can move to the next. Taking a slice of this hemisphere at some constant altitude, and removing any sections in which the visibility of the target is blocked by obstacles, will create a visibility polygon. The application of a constraint on turning radius, which dictates constant velocity, completes the description of the problem. Considering each of these visibility polygons must be visited by a UAV, these polygons are allowed to overlap, a minimum turning radius constraint can be similar as restricting as desired, and that multiple UAV's starting from multiple depots must cooperatively solve the problem, this scenario is very complex.

A brief overview of the genetic algorhtm (GA) SCROOGE will be presented. Given a random set of solutions, each solution represented by a "string", a GA then follows evolutionary patterns to find a near-optimal solution to some cost function. Here a string consists of an array of numbers representing the order of targets visited.

[1] Graduate Student, School of Aerospace Systems, University of Cincinnati, AIAA Student Member
[2] Associate Professor, School of Aerospace Systems, University of Cincinnati, AIAA Associate Fellow

GA's traditionally utilize crossover (two parents creating an offspring which has traits of both) and/or mutation (some form of random small transformation in an offspring, such as switching a single value from 0 to 1). Due to the special constraints of the TSP, traditional crossover cannot be utilized, as breeding of two parent strings could cause an offspring route to visit a city twice. However, there are methods for a GA to work around this problem. One of the simplest and most computationally efficient is found by replacing traditional crossover with self-crossover (Kundu and Pal [4]).

The breeding mechanisms present in this algorithm, a combination of self-crossover and mutation, can be seen in Table (1).

| Parents | Crossover | Self-Crossover | Mutation | Self-Crossover + Mutation |
|---|---|---|---|---|
| **1 2 3 4 5 6** | 1 2 3 : 3 2 1 | 5 6 : 3 4 : 1 2 | 1 2 **4 3** 5 6 | 5 6 : **4 3** : 1 2 |
| **6 5 4 3 2 1** | | 2 1 : 4 3 : 5 6 | 6 **3** 5 **4** 2 1 | 2 **3** : 4 **1** : 5 6 |

**Table 1. Example GA reproduction methods for a six city TSP tour.**

Utilizing a tournament polling style and parameters that morph as the run enters three distinct stages (initial, intermediate, and final), SCROOGE has been shown to have good performance [5].

Described in detail in [6], the Fuzzy Logic System (FLS) "UNCLE" takes each targets (or in this case, the centroid of each polygon) coordinates and translates them to polar coordinates with respect to the depot being analyzed. Fuzzy c-means clustering is then utilized on the angle values of these coordinates. Fuzzy c-means clustering, a variation on MacQueen's K-means clustering [7], takes a set of coordinate measures and forms a given number of clusters based on the minimization of the following cost function [8]:

$$J_m = \sum_{i=1}^{N}\sum_{j=1}^{C} u_{ij}^{m}\left\|x_i - c_j\right\|^2, \quad 1 \le m < \infty$$

$$u_{ij} = \frac{1}{\sum_{k=1}^{C}\left(\frac{\left\|x_i - c_j\right\|}{\left\|x_i - c_k\right\|}\right)^{\frac{2}{m-1}}}, \quad c_j = \frac{\sum_{i=1}^{N} u_{ij}^{m} \cdot x_i}{\sum_{i=1}^{N} u_{ij}^{m}}$$

Here "u" is the degree of membership, and "c" is the center of the cluster, for "m" clusters. This extension of K-means clustering allows each point to belong to multiple clusters. The output in the 5 UAV case will be the coordinates of the 5 cluster's centers, and the membership values that each point has to these clusters.

While this breakdown of the points into a separate subset for each UAV is an effective technique, improvements were noticed when a fuzzy logic system was implemented to take each point's membership into these clusters as an input.

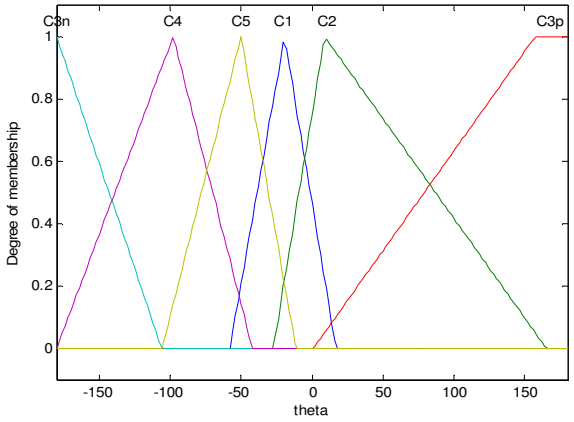


**Figure 1: 5 UAV UNCLE input membership functions for random target distribution**

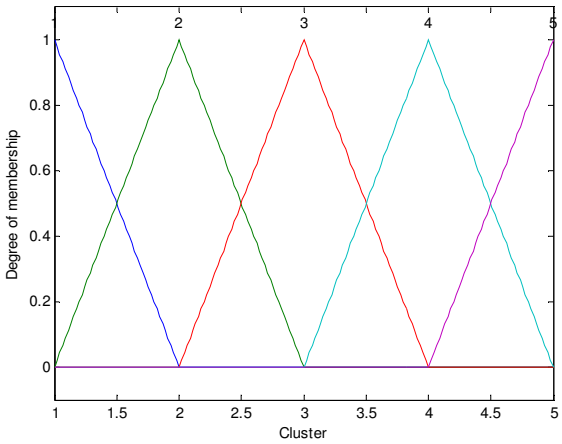


**Figure 2: 5 UAV UNCLE output membership functions for random target distribution**

Each input membership function's peak is a center of a cluster, and each membership function overlaps the center to its left and its right by a set amount. These figures are for a 5 UAV case; the membership functions change with each run. This set overlap was shown to provide good performance, and is a very cost effective method, as shown in [6]. While originally the membership functions were defined based on the density of targets within the sweep of the cluster's angle, this was previously shown to be costly and, sometimes, inaccurate. By simply using this set overlap of membership functions, which can be seen in Figure (1), the system was autonomous and results found to be more accurate and drastically quicker. The rules and outputs for this system are incredibly simple, where if an input belongs mainly to cluster 1 ("C1"), that point is put with UAV 1, and so forth. They are shown below:

- If (Input is C1), then (Output is UAV1)
- If (Input is C2), then (Output is UAV2)
- If (Input is C3 Positive or C3 Negative), then (Output is UAV3)
- If (Input is C4), then (Output is UAV4)
- If (Input is C5), then (Output is UAV5)

For cases where the targets do not fully surround the starting location, or depot, only 5 membership functions are utilized, as there is no need for jumps between -180 to 180 degrees.

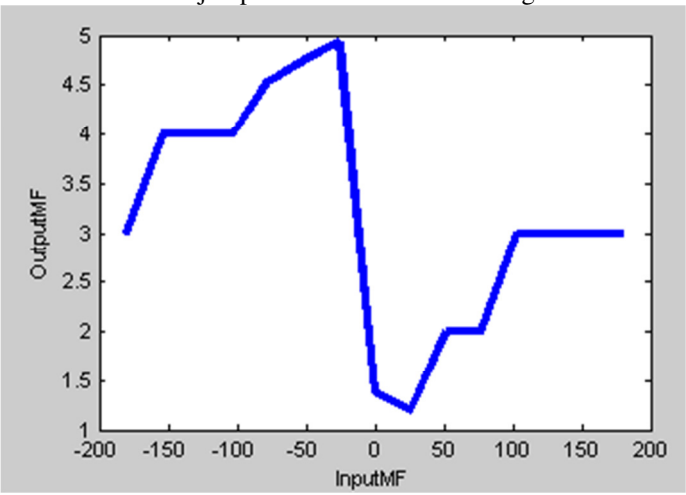


**Figure 3: 5 UAV UNCLE rule surface for random target distribution**

This rule surface was unexpected; it is a product of utilizing different fuzzy logics to cluster the cities. The refinement of this system, which follows simplistic steps but provides very effective results, took place over

multiple iterations. This was accomplished by noticing the improvements at each step, as shown in a previous publication [6].

The defuzzification method, or the way in which a fuzzy result is released as a crisp output from the system, is the middle of maxima (MoM) method. Here, a control action is made based upon the average value or all local inputs that reach the maximum membership. Avoiding the centroid defuzzification method furthered the difference between the fuzzy c-means clustering and the fuzzy logic system UNCLE is based upon. Additionally, wider membership functions, such as Cluster 3 in Figure (1) can sometimes dominate the border between clusters. This is due to the fact that if the centroid defuzzification method is utilized, a smaller level of membership in one function can outweigh a higher membership in the function next to it due to a smaller base. This method produces a result where the border from one cluster to another is not as pre-defined, thus avoiding some scenarios where a suboptimal target distribution is noticed.

These two techniques when combined have shown great effectiveness in solving larger-scale MTSP's [6], however their application to the MDPVDMTSP truly showcases their strengths.



## II. Application to the MDPVDMTSP

Through a series of additions to UNCLE SCROOGE, a multitude of functions were created that broke down each section of this problem, in a manner as to avoid introducing error whenever possible.

Just as a MTSP is broken down into multiple TSP's, the multi-depot MTSP (MDMTSP) is reduced to multiple MTSP's. This is accomplished through another clone of the UNCLE system, which utilizes the exact same process, except remaining in the Cartesian coordinate system.

This results in an even distribution of waypoints if the depots are both located towards the middle of the map. Such a distribution more closely resembles the time-optimal MDMTSP. If one depot is located in a corner behind another depot, it will only receive the minimum number of points so each aircraft can satisfy its minimum tour length requirement. This is more of a distant optimal MDMTSP. As the TSP is the optimal distance solution for any MTSP, so too is the MTSP to a MDMTSP. Clearly if given a specific objective (time or distance optimizing) the cost function would need to be altered to either optimize towards average UAV tour length (time-optimal MDMTSP for constant velocity) or for total combined UAV flight distance.

A much more complex addition to the system is the capability to change from the base TSP to the Polygon-Visiting Dubins TSP (PVDTSP). First, given a set of polygons, the centroids of each are calculated. These centroids are sent to SCROOGE to run through the GA, producing a result such as:

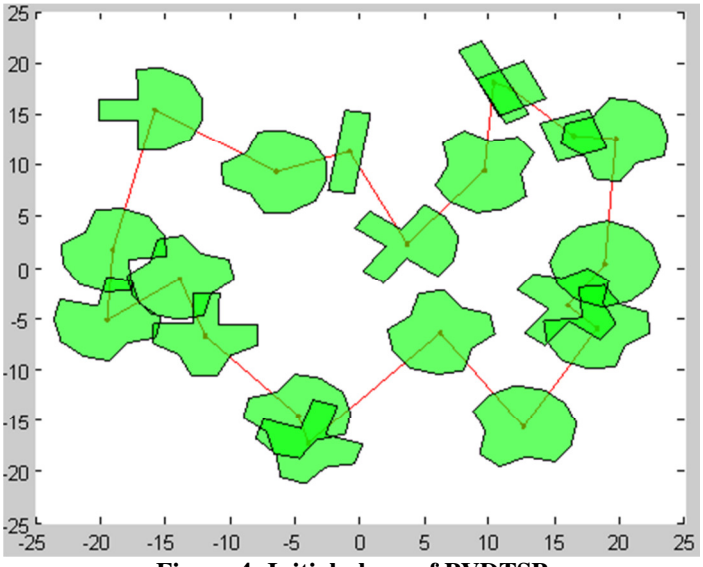


**Figure 4: Initial phase of PVDTSP**

The above is simply utilized to determine what order the polygons are visited in, not necessarily where in the visibility polygon the UAV crosses. It is important to note that extremely irregular polygons, especially ones that

are very elongated, may produce error in determining the order of polygons to visit. With the current definition of a visibility polygon being a plane of a hemisphere with possible blocked portions, such a polygon will not occur.

Following this, the point along the border of the polygon that the UAV actually crosses must be determined. Breaking this down effectively using a simplified form of dynamic programming allows the program to utilize brute force, but still keep a satisfactory run-time. Instead of simultaneously examining every polygon, the problem looks are three polygons at once. Let i be the polygon that is being examined, i-1 be the polygon before it in the string, and i+1 the one after t.

Initially, the point in which the code evaluates the polygon from is the centroid. Then the following cost function is calculated for each point, n, sampled along the border of the $i^{th}$ polygon:

$$Cost = Distance + AngleFactor * (180° - Angle)$$

Where "AngleFactor" is just some weight applied to the Angle value to increase its importance in the cost function and:

$$Distance = \sqrt{\left(\left(y_{i,n} - y_{i-1}\right)^2 + \left(x_{i,n} - x_{i-1}\right)^2\right)} + \sqrt{\left(\left(y_{i+1} - y_{i,n}\right)^2 + \left(x_{i+1} - x_{i,n}\right)^2\right)}$$

$$Angle = \arccos\left(\frac{a^2 + b^2 - c^2}{2ac}\right)$$

$$a = \sqrt{\left(\left(y_{i+1} - y_{i,n}\right)^2 + \left(x_{i+1} - x_{i,n}\right)^2\right)}$$

$$b = \sqrt{\left(\left(y_{i,n} - y_{i-1}\right)^2 + \left(x_{i,n} - x_{i-1}\right)^2\right)}$$

$$c = \sqrt{((y_{i+1} - y_{i-1})^2 + (x_{i+1} - x_{i-1})^2)}$$

These equations provide the distance of the two line segments that the 3 points create, and the angle they make with the $i^{th}$ polygon as the vertex. The figure below depicts this process:

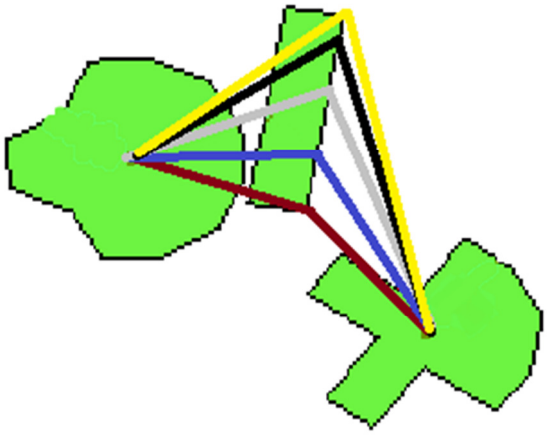

**Figure 5: Polygon Point Selection Process**

The point that produces the shortest and straightest set of lines would be considered optimal. A higher weight on the angle will take a longer path that produces less turning. A balance is required when Dubins paths are added. Two measures of this cost are recorded; static, where only the three polygons are examined, and stacking, were the cost is additive around the entire loop.

Each iteration updates the point selected on the $i^{th}$ polygon; for the first iteration this means switching from the centroid t some point on the border. The process continues a set number of iterations, and does so at a much quicker time than analyzing all polygons at once. It is important to note that a check is necessary to determine if the point chosen is inside another polygon, in which case that polygon can be skipped.

The heading at each point must now be found, to find a pose (x, y, θ) that can be fed as an input to a Dubins path solver. The code simply determines these θ's by diving the angle utilized by the preceding cost function in half. This results in a well-spread distribution of turning between the polygons. After finding the order of polygons visited, and the corresponding series of poses (x, y, θ), the PVDTSP is solved by simply running through a Dubins path solver. The particular one utilized in this study (thanks again to Steve Rasmussen at AFRL) takes a starting pose and an end pose, and find the optimal path through a series of either turn-straight-turn, turn-turn-turn. straight-turn, or turn-straight maneuvers depending on the headings and the set minimum turning radius. The image below, courtesy of Steven Lavalle at Cambridge University [9], shows some example poses, q, and their appropriate Dubins paths.

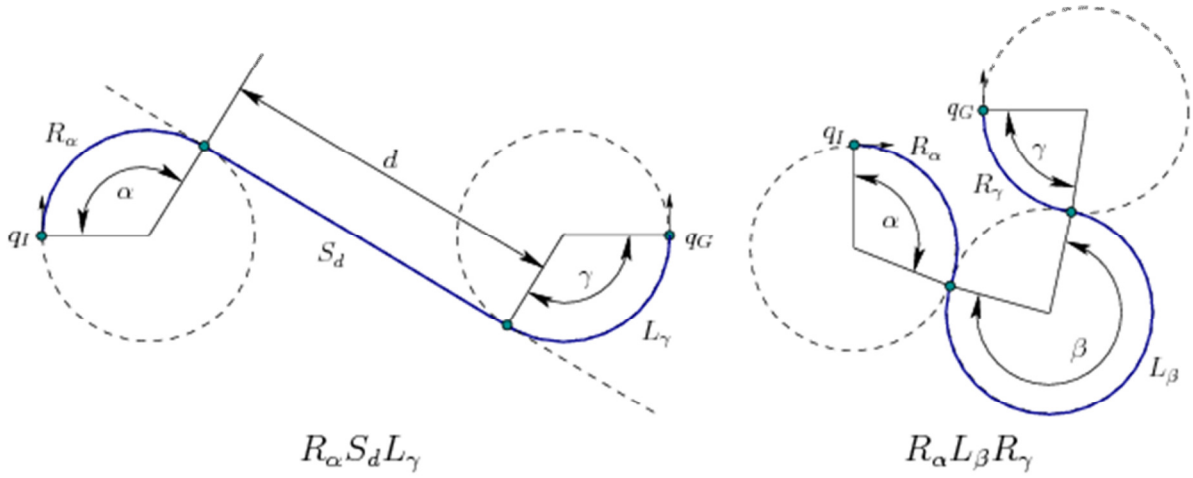


**Figure 6: Example Dubins Paths [9]**

Combining these methods together, UNCLE SCROOGE can been adapted to solve the more complex MDPVDMTSP. A 3 depot, 12 UAV (4 at each depot) MDPVDMTSP run is broken down as follows:

- Modified UNCLE system distributed polygons to each depot
- Each depot becomes a MTSP, where the UNCLE system is run based on centroids of polygons
    - A cluster created by UNCLE is assigned to a particular UAV
    - Now a collection of PVDTSP's, SCROOGE is ran for each based on centroids
        - Afterwards, points are selected on each polygon based on the iterative process described previously
        - Poses are created and run through the Dubins solver, with a set minimum turning radius, to calculate distance
    - The code runs through this process for each UAV at the depot
- The sum of the solutions from each PVDMTSP problem is the solution to the MDPVDMTSP

For the capability to handle any polygon count, with up to 4 depots, and up to 5 UAV's at each depot, the code is a combination of 34 Matlab function files, with one master .m file, totaling to over 10,000 lines of code. The amount of heuristic methods and assumptions present is certainly more than other approaches; something that does not sit well with critics of such techniques.

During the infancy of this research, the code for the genetic algorithm SCROOGE was simple enough to run trade studies on Monte Carlo simulations with different values for each parameter. Polling size, self-crossover rate, mutation rate, and population size were the only variable parameters. Coupling is present between these, but with this short list, three iterations of trade studies brought convergence [5]. However, this only looked into set intervals, and only two decimal places for the percent parameters.

Such a method would not be feasible given the time of this research for the code described in this paper. The following parameters needed to be determined:

| Parameter | Description |
|---|---|
| C_r1 | Crossover rate for SCROOGE during Initial Phase |
| C_r2 | Crossover rate for SCROOGE during Intermediate Phase |
| C_r3 | Crossover rate for SCROOGE during Final Phase |
| M_r1 | Mutation rate for SCROOGE during Initial Phase |
| M_r2 | Mutation rate for SCROOGE during Intermediate Phase |
| M_r3 | Mutation rate for SCROOGE during Final Phase |
| P_1 | Tournament polling size for SCROOGE during Initial Phase |
| P_2 | Percent value of P_1 that represents polling size during Intermediate Phase |
| P_3 | Percent value of P_1 that represents polling size during Final Phase |
| gens | Scalar that is multiplied by Polygon count to determine how many generations to run each instance of SCROOGE for |
| pop | Scalar that is multiplied by Polygon/city count to determine how many strings are utilized in each instance of SCROGE |
| O | Utilized by UNCLE to determine how much the membership functions overlap |
| AngleFactor | Scalar that is multiplied by the angle found in polygon point selection; puts more weight on ease of turning rather than optimizing by distance |

**Table 2: Variable Parameters within UNCLE SCROOGE**

A 13 parameter trade study, with coupling present amongst many of them being out of the question, an alternate method was investigated. Reviewing the strengths of genetic algorithms, it is recalled that the only information necessary is a cost function and the constraints on feasible solutions. As such, another genetic algorithm deemed "GOGS" (Genetic algorithm Optimizing Genetic algorithmS) was created. Here the strings consisted of 13 numbers that ranged from 0 to 1, up to 4 decimal places (the result of the Matlab "rand" function). GOGS' cost function is UNCLE SCROOGE. That is, each string is a series of inputs that are run through the code, with some pre-determined scenario. For the percentage-based parameter, no manipulation of these values was necessary. For each of the other parameters (P_1, gens, pop, O, and AngleFactor), an upper bound was chosen. The percentage is multiplied by this upper bound before being sent as an input.

Due to the randomness present in UNCLE SCROOGE, GOGS evaluates each string three times, and utilizes the average of these values as the fitness of each string. This genetic algorithm utilizes a combination of traditional crossover and mutation. There is only one mutation mechanism present, where a random member of the string is replaced by a new random number. Running GOGS was a lengthy optimization process, but certainly faster than random or brute searches.

GOGS was run on a Windows 7 based desktop, with 16.0 GB of RAM (4GB x 4) and an Intel 3.4 GHz dual core processor. As this code does not utilize parallel processing, this machine was more suited for the lengthy task. 25, 50, 100, 150, and 200 polygon cases were utilized, all with 1 depot and 5 UAV's, as there are no variable parameters

for the multi-depot section. While it is doubtful that the true optimal values for each parameter was found, performance was noticeably improved after the development and running of GOGS.

.

## III. Results

The work and results shown in this paper were constructed with the Matlab programming language, on a Windows based laptop with an Intel i7 1.73 GHz quad-core processor and 6.00 GB (2GB x 3) of RAM. Comparison opportunities for this work are limited; however performance can be benchmarked against Obenmeyer for a 20 polygon PVDTSP case. Results for UNCLE SCROOGE being applied to the MDPVDMTSP will be shown and discussed as well.

As the first work investigating this particular variant of the TSP, Dr. Obenmeyer's research [3] serves as the only current basis for comparison between these methods. His work examines a few particular cases, but we will only compare the most complex case here: a 20 polygon map that is tightly packed, includes some overlapping polygons, and has a relatively small minimum turning radius of 3 meters (though these units are arbitrary). Many methods would find difficulties with this layout, resulting in 360 degree maneuvers and U-turns.

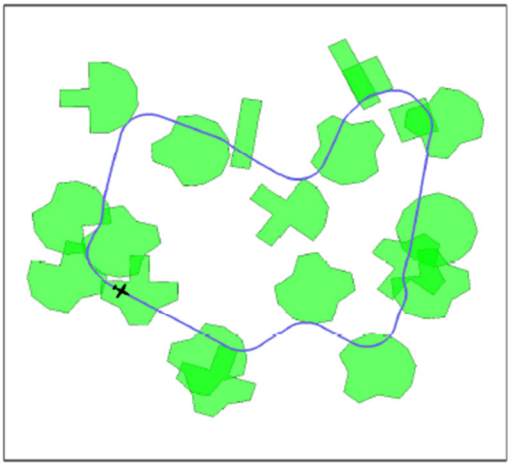

**Figure 7: 20 Polygon PVDTSP [3]**

It is important to note that the above figure also depicts Obenmeyer's best found solution of 118.99 meters. UNCLE SCROOGE's solution is found below:

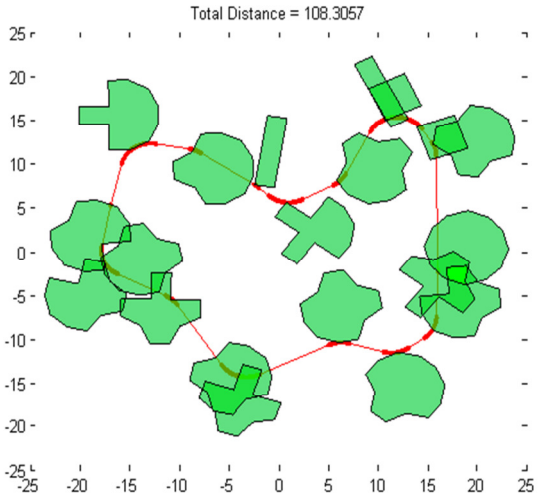


**Figure 8: UNCLE SCROOGE's Solution to Sample PVDTSP**

Graphically it is somewhat difficult to tell these solutions apart, but there are a few differences. These differences are exemplified in the following table, though a vital preliminary note must be discussed. While his work presents the above solution as a "1500 sample Resolute Complete" run, his data also shows that he received a similar result in a roughly "1125 sample Resolute Complete".

| Code | Optimal Distance (m) | Computational Cost (sec) | Chance of obtaining optimal |
|---|---|---|---|
| **1500 Sample Resolute Complete** | 118.99 | 506.07 | 100% |
| **Roughly 1125 Sample Resolute Complete** | 118.99 | Roughly 175 | 100% |
| **UNCLE SCROOGE** | 108.31 | 17.09 | 99% |

**Table 3: Comparison between "Resolute Complete" roadmap method and UNCLE SCROOGE**

As seen in Table (3), whether comparing to the strongest Resolute Complete method or the first to present the best solution, UNCLE SCROOGE can find a much better result in much less time. This is despite the fact that the Obenmeyer's results are found on a computer that is slightly more optimized for such tasks (again, UNCLE SCROOGE does not utilize parallel processing), but is also created in C++ [3]. Matlab, while praised for its user-friendliness and ease of code creation, is known to be a slow-running computer language. Conversion of UNCLE SCROOGE to C++ would indeed reduce the run time. By what degree is unknown offhand, but due to the amount of loops in UNCLE SCROOGE, it ought to be quite noticeable.

Comparing the two routes, it is clear that the polygon border point selection algorithm in UNCLE SCROOGE is operating well. Also note that the order of polygons visited for both solutions seems to be the same. For this case, the centroid assumption to utilize SCROOGE allows rapid run-times with no loss of performance. Additionally, while the code utilizes methods that produce uncertain results, over the average of 100 runs a 99% optimality rate was found with an average run-time of 17.09 seconds.

The results show the drastic increase in performance and speed that can be received by adopting approximating control techniques, if one can stomach a 1% chance of not obtaining the optimal solution. Technically speaking, if UNCLE SCROOGE's solution of 108.31 meters is the current best solution to this problem, or the current optimal, the Resolute Complete methods have a 0.00% chance of obtaining the current optimal value. If the small chance of sub-optimality is a reason for one to be uneasy, this should help cope with the risk.

Now examine the results if the minimum turning radius is pushed even further, say from 3 meters to 4. A solution without any U-turns or 360 degree maneuvers is certainly feasible if the route is pushed back to the far external edges of the outside polygons. Here is what UNCLE SCROOGE outputs:

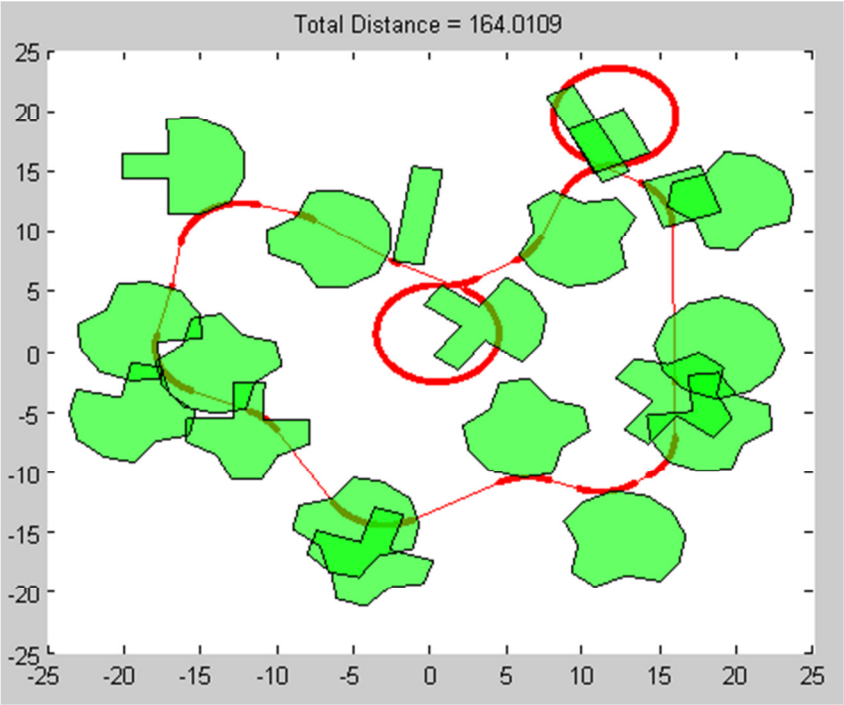


**Figure 9: Increased Minimum Turning Radius Solution**

The code could not compensate for such a tight turning maneuver and required some long turning maneuvers. How much off from optimal this solution is currently unknown, but it is clear that improvements for extremely tight turning radii cases are possible.

Now the final results will be covered for the end-goal of this research, solving the Multi-Depot Polygon-Visiting Dubins Multiple Traveling Salesman Problem. As far as what can be publically found, such a problem has not been examined previously. However, we can compare these results performance-wise with the final aim of developing a real-time target allocation and path planning algorithm. This entails that runtimes must be similarly low and scalability must be satisfactory. For these cases, extremely large minimum turning radii relative to polygon size and distribution will not be examined, as UNCLE SCROOGE's strengths in this area are shown in the PVDTSP comparison.

First to be presented is a relatively simple case, a 50 Polygon, 2 depots, 1 UAV/Depot MDPVDMTSP. The figures in this section will not show the actual curvature of the UAV's path, but rather the points at which it begins a turning maneuver. Solutions to a few random cases with these parameters are shown.

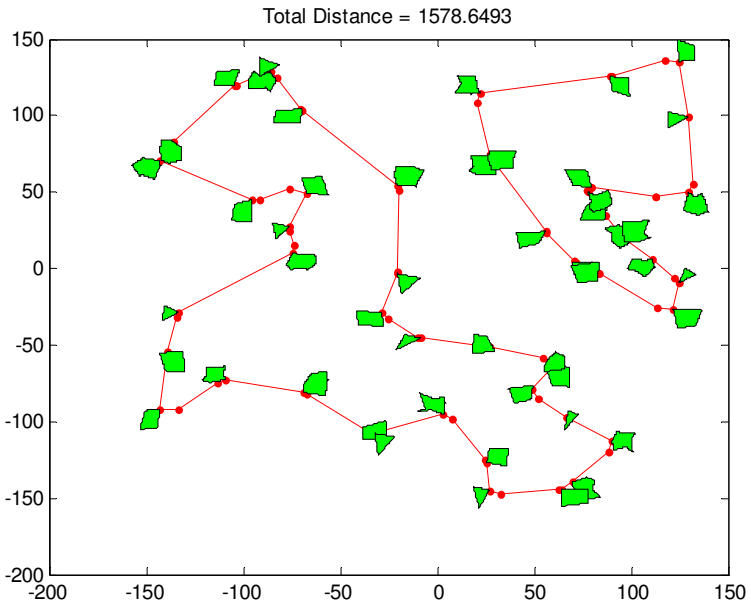


**Figure 10: 100 Polygon, 2 Depot, 1 UAV/Depot Example 1**

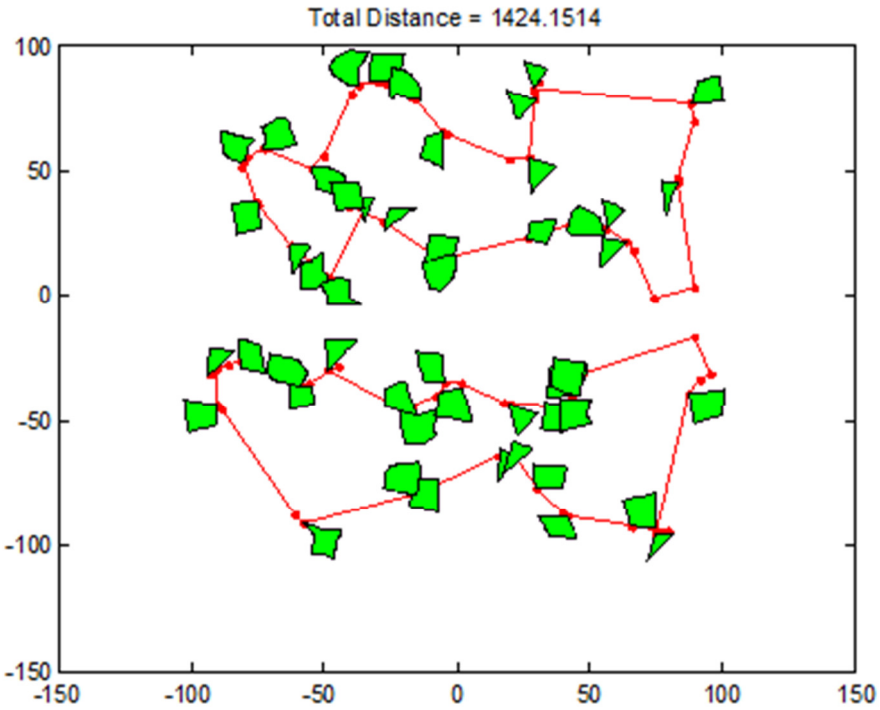


**Figure 11: 100 Polygon, 2 Depot, 1 UAV/Depot Example 2**

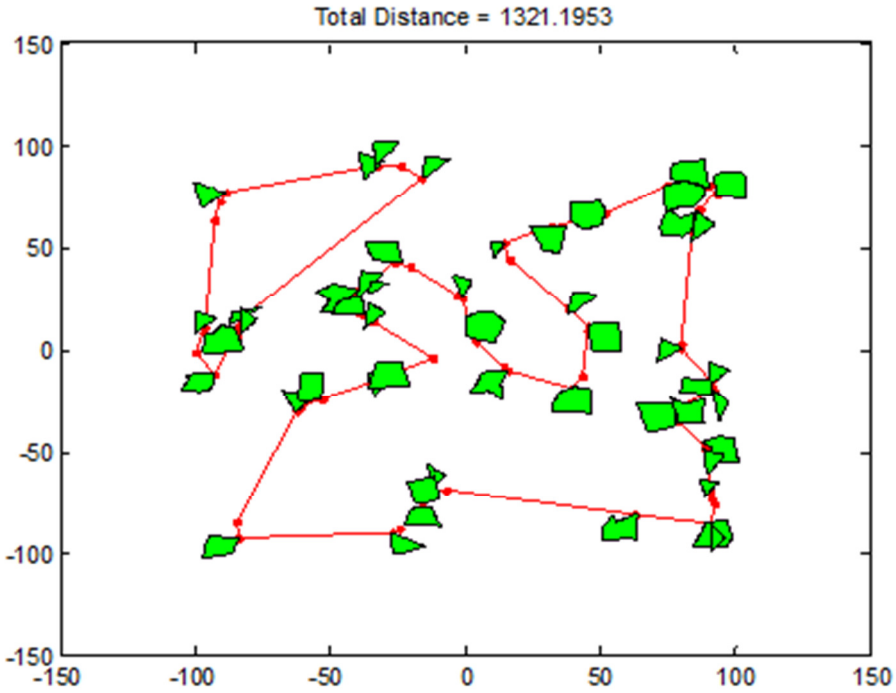


**Figure 12: 100 Polygon, 2 Depot, 1 UAV/Depot Example 3**

An average of 100 runs resulted in a run-time of 12.27 seconds for these cases. With a less strict minimum turning radius, run-time is reduced compared to the tests ran in 5.1. These cases more closely resemble a desert observation patrol, a much looser environment in terms of maneuverability. The possible distribution issue with the modified UNCLE system to break down the MDPVDMTSP into two different PVDMTSP's is shown in Figure (12). Here, one depot is located in the back corner, and since distance optimization is the priority, takes a much less dominant role than the central depot.

Moving on to a more complex case, the capabilities of the current UNCLE SCROOGE are maxed with 4 depots, 5 UAV/depot simulations. First, smaller-scale cases will be examined; results will be shown with 250 polygons below:

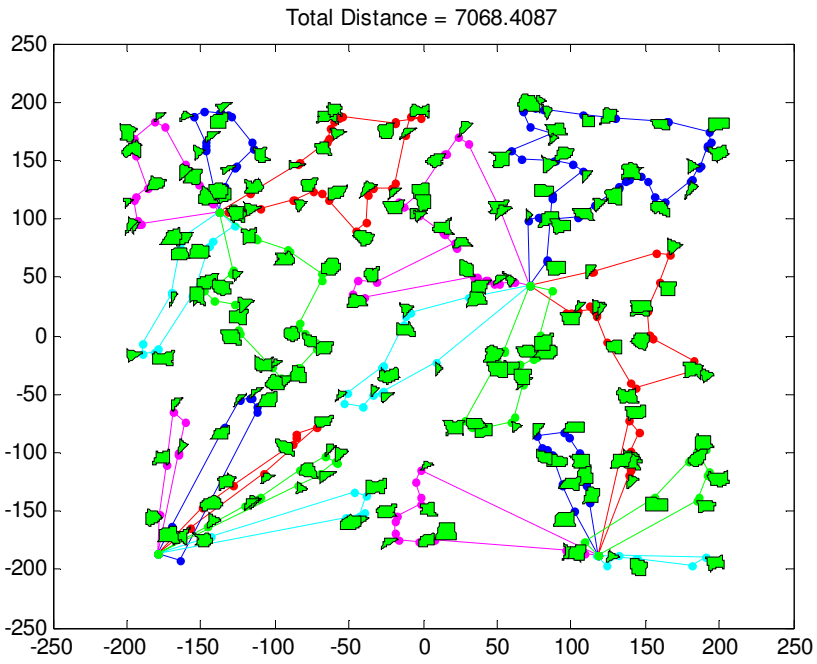


**Figure 13: 250 Polygon, 4 Depot, 5 UAV/Depot Example 1**

In the example shown in Figure (13), 2 depots were located along the bottom edge of the map; a good display of UNCLE SCROOGE's freedom of depot placement. Depots do not have to be located within the center of the polygon field. Here each UAV had to visit a minimum of 3 polygons, with the upper depots covering a larger area.

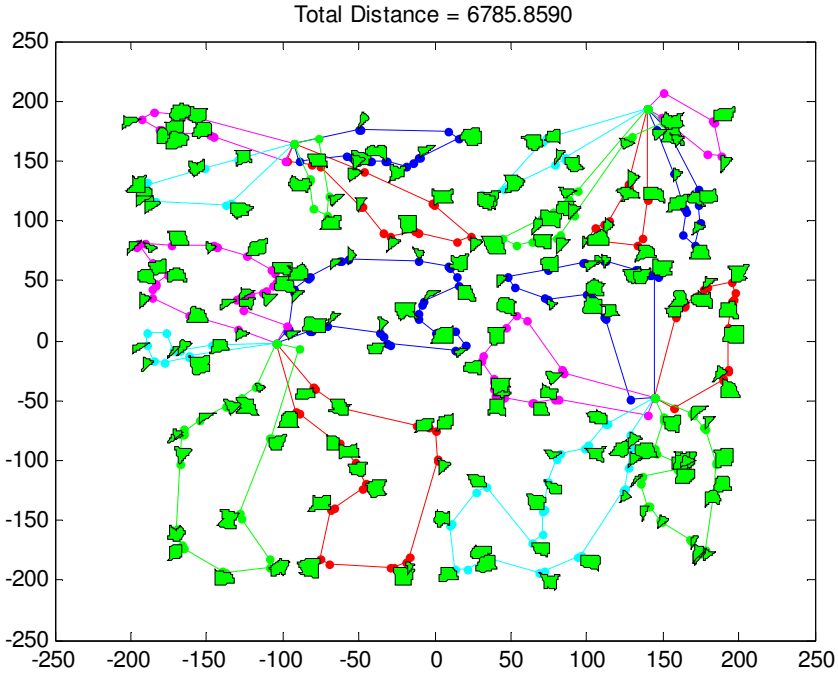


**Figure 14: 250 Polygon, 4 Depot, 5 UAV/Depot Example 2**

Figure (14) shows a slightly more balanced distribution. While there is no method to determine what percent optimal these runs are, by inspection, and based on performance of the individual PVDTSP solver, they appear to be accurate. There are clearly a few polygons on the border of each cluster that could improve overall fitness by switching to a neighboring cluster, though these are the rarity.

Over a similar 100 runs, an average run-time of 25.62 seconds was found though with more variability. This is understandable, as particularly long routes for individual UAV's will introduce more computational cost than a more balanced set of routes. Considering that initially, the goal for SCROOGE was to solve 100 city TSP's in under 30 seconds [5], the fact that the code can now solve 250 polygon, 5 depot, 20 UAV MDPVDMTSP problems in a similar timeframe is very promising.

Lastly, a large-scale case was examined to get a feel for scalability through such demanding tests. Presented below is the solution UNCLE SCROOGE with similar parameters, except for larger polygon counts

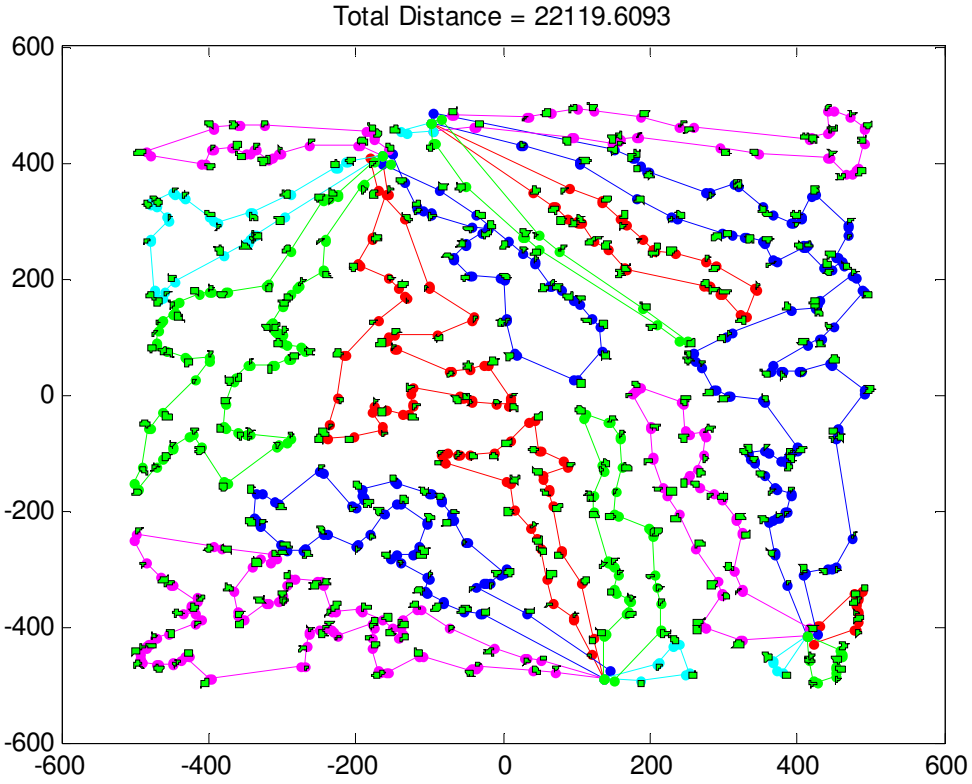


**Figure 15: 500 Polygon, 4 Depot, 5 UAV/Depot Case**

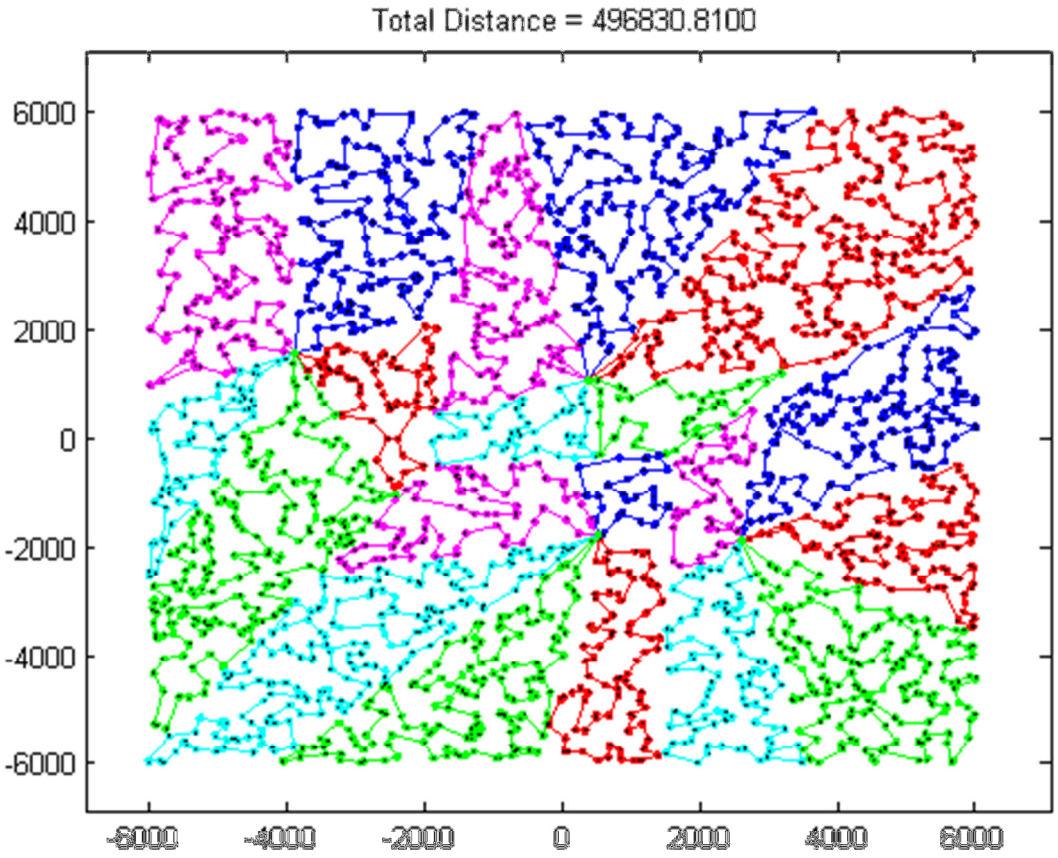


**Figure 16: 2500 Polygon, 4 Depot, 5 UAV/Depot Case**

This last case may not be encountered by any modern UAV swarm, but provides insight into UNCLE SCROOGE's performance. Taking 35,357 seconds, or 9.8 hours, it could hardly be imagined to be implemented into a real-time controller, but does show the scaling capabilities of the algorithm.

Based upon these trade studies of 100 runs each, except for the 2,500 polygon case, Table (4) shows the time-performance of UNCLE SCROOGE for the sample cases.

| Case | Mean Run-time (sec) | Standard Deviation (sec) |
|---|---|---|
| 20 Polygon PVDTSP | 17.09 | 0.91 |
| 100 Polygon MDPVDTSP | 12.27 | 0.79 |
| 250 Polygon MDPVDMTSP | 25.62 | 2.82 |
| 500 Polygon MDPVDMTSP | 261.33 | 19.64 |
| 2500 Polygon MDPVDMTSP | 35357 (9.8 hours) | N/A |

**Table 4: Comprehensive Run-Times for Sample Cases**

## IV. Conclusions and Recommendations

This research developed a method that hybridizes genetic algorithms, fuzzy logic systems, and, to an extent, dynamic programming. Through these powerful optimization tools, the PVDTSP presented by Dr. Obenmeyer has been expanded. Without any benchmarks for the MDPVDMTSP, UNCLE SCROOGE explored into this new scenario, which more closely represents a UAV swarm cooperative control problem.

While constant velocity and altitude are unrealistic constraints, this is variant of the MTSP is approaching functionality in a realistic case. UNCLE SCROOGE was compared against the techniques of Obenmeyer, and then

developed to solve the more complex problem. UNCLE SCROOGE was found to have a 9.8% increase in accuracy, as well as a decrease of run-time surpassing 1000% or a factor of 10. This was despite operating on a slightly slower computer, and in a slower programming language. This increase in performance does however come at the cost of only having a 99% chance of reaching this optimal answer.

In solving the MDPVDMTSP, it was found that in less than 30 seconds, cases with 4 depots, 5 UAV/depot, and up to 250 polygons were solvable. Table (4) shows that while this code does not scale linearly with time, it still performs well. Additionally, large cases such as the 2,500 polygon example are solvable, taking 9.8 hours for this particular case. The sheer quantity of possible solutions for this problem is staggering, many times more than a 2,500 city TSP.

Some weaknesses of the code are noticeable. Depots that are located within corners receive very little use compared to those in the middle of a map. Avoiding this issue requires clarification of the cost functions utilized. When other methods present themselves for solving the MDPVDMTSP, this clarification can be determined for benchmarking purposes.

Additionally, the modified Cartesian UNCLE system in charge of clustering the depots shows occasional weakness. Some polygons on the borders of a cluster would bring an overall decrease in cost if switched to a neighboring cluster. UNCLE however occasionally fails to allow this due to distance of the polygon's centroid to the center of the cluster. If additional UAV's/depot, or depots are desired, they can be implemented easily enough. This time consuming process was not deemed necessary until other methods approach that this can be tested against. Being the variant of an only-recently studied problem, the MDPVDMTSP is a difficult test of an algorithms performance. UNCLE SCROOGE has shown great strength in solving the PVDTSP, as well as the MDPVDMTSP.

The capability to solve such a problem as the MDPVDMTSP lends great promise to a code's ability to solve cooperative control problems faced by UAV swarms. Initially, due to the sheer size and complexity of the code, further performance optimization is possible. Additional mechanisms for the genetic algorithm, logics to prevent long maneuvers in difficult minimum turning radius PVDTSP's, and extra honing of the variable parameters could all bring about noticeable increases in performance. A prime example of an possible area of improvement is developing an efficient coding for utilizing the Dubins path solver with the polygon-boundary point selection logic. This would refine the solution space, likely bringing improvements in both accuracy and run-time. While the fuzzy logic system UNCLE currently is single input, additional inputs could create a better cluster distribution. Bringing additional control techniques into the code may also strengthen it, for example utilizing methods to lower the possible number of strings the genetic algorithm sees, thus reducing run-time.

Additional optimization will most likely always be present, but currently UNCLE SCROOGE is ready to approach further complexities. Removing the constant height altitude switches the problem to three-dimensional. This could be implemented easily enough with some minor adjustments throughout the code and the development of new cost functions. The new cost functions would have to include basic aircraft dynamics in terms of altitude increase costs and altitude decrease gains of energy. Complete aircraft dynamics, and a transfer from a static to a dynamic problem would be necessary to create a fully-usable real time controller, but perhaps this is not the optimal direction of UNCLE SCROOGE. While investigation into this may provide similar strong results as this study, it may also be that UNCLE SCROOGE should work in tandem with other control programs. If this is to be utilized real-time, the genetic algorithm engine must be able to handle a change in number of targets or obstacles mid-run, meaning that the string size must be able to change between generations. Additional research into this area would determine the feasibility of utilizing the program in this manner.

Another complexity that can be added is obstacles. This could model no-fly zones, mountainous regions, and other large-scale obstructions faced by full-sized UAV's. For micro-UAV's, obstacles could simply be the walls inside a structure, or buildings alongside a city street. For use with police forces, urban micro-UAV's would also require factoring in the wind between these stretches of buildings. Without adding a complete set of aircraft dynamics, velocity may be allowed to be adjusted by determining minimum turn radius of the aircraft at each velocity. Cost functions would then have to reflect the added cost in acceleration time afterwards.

UAV's are getting more economical and accessible. The removal of a pilot, or as with a UAV-swarm, multiple pilots, brings the same affordability and ease of implementation to a UAV-swarm. Law enforcement, search and rescue, and firefighters could utilize such systems much easier without the added cost of trained pilots.

While in the future this may mean the ability to create a massive army of attack drones at the rate our factory can pump them out, there are more realistic benefits. This work could be applied to assist with nano-machine navigation. A team of robotic probes can be sent on space missions, to distances outside of reasonable control time. While the ability to acquire data about their surroundings would be necessary, the end product of this work would allow them to autonomously travel to their destination, avoiding risk and taking time-optimal routes.
The immediate future of this research includes a combination of adding these additional complexities to the problem statement and optimizing the existing algorithms. The potential for this work is high, and a finalized product would be valuable to a wide variety of industries. While this work has utilizes a multiple popular techniques, it has become unique on its path towards effective UAV swarm cooperative control through application and utilization of these methods. However, there is still a great deal more work to be done before it can be properly utilized. Further refinement of the existing methods and investigation into additional processes will allow this work to be applied in a variety of areas.



## V. References

1. "Rise of the Machines: UAV Use Soars", Military.com Associated Press, 2008 http://www.military.com/NewsContent/0,13319,159220,00.html

2. Osborn, K., "Army surpasses 1 million unmanned flight hours", 2010 http://www.army.mil/article/38236/army-surpasses-1-million-unmanned-flight-hours/

3. K. J. Obermeyer. Visibility Problems for Sensor Networks and Unmanned Air Vehicles. PhD Dissertation, Department of Mechanical Engineering, University of California, Santa Barbara, June 2010.

4. Malay K. Kundu and Nikhil R. Pal. "Self-crossover and its application to the traveling salesman problem." Multiple Approaches to Intelligent Systems, 12th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE-99), volume 1611, pages 326–332, Cairo (Egypt), 1999. Springer-Verlag.

5. Ernest, N., and Cohen, K., 2011, "Self-Crossover Based Genetic Algorithm for Performance Augmentation of the Traveling Salesman Problem", AIAA, Infotech@Aerospace 2011, St. Louis, Missouri.

6. Ernest, N., and Cohen, K., 2012, "Fuzzy Logic Clustering of Multiple Traveling Salesman Problem for Self-Crossover Based Genetic Algorithm", AIAA, 50th ASM 2012, Nashville, TN.

7. MacQueen, J. B., "Some Methods for classification and Analysis of Multivariate Observations, Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability", 1967, Berkeley, University of California Press, pp. 281-297.

8. Dunn, J.C., "A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters", 1973, Journal of Cybernetics, pp. 32-57, http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/cmeans.html

9. LaValle, S.M., "Planning Algorithms: Dubins Curves", Cambridge University Press, 2006, Section 15.3.1, http://planning.cs.uiuc.edu/node821.html