University of Cincinnati

Date: 5/9/2012

I, Nicholas D Ernest , hereby submit this original work as part of the requirements for the degree of Master of Science in Aerospace Engineering.

It is entitled:

UAV Swarm Cooperative Control Based on a Genetic-Fuzzy Approach

Student's name: **Nicholas D Ernest**

This work and its defense approved by:

Committee chair: Kelly Cohen, PhD

Committee member: Manish Kumar, PhD

Committee member: Bruce Walker, ScD

UNIVERSITY OF Cincinnati

2660

UAV Swarm Cooperative Control

Based on a Genetic-Fuzzy Approach

A thesis submitted to the
Graduate School
of the University of Cincinnati
in partial fulfillment of the
requirements for the degree of

Master of Science

in the Department of Aerospace Engineering
of the College of Engineering

by
Nicholas D. Ernest

B.S. University of Cincinnati

May 9th, 2012

Committee Chair: Kelly Cohen, Ph.D.

ABSTRACT

The ever-increasing applications of UAV's have shown the great capabilities of these technologies. However, for many cases where one UAV is a powerful tool, an autonomous swarm all working cooperatively to the same goal presents amazing potential. Environment that are dangerous for humans, are either too small or too large for safe or reasonable exploration, and even those tasks that are simply boring or unpleasant are excellent areas for UAV swarm applications. In order to work cooperatively, the swarm must allocate tasks and have adequate path planning capability.

This paper presents a methodology for two-dimensional target allocation and path planning of a UAV swarm using a hybridization of control techniques. Genetic algorithms, fuzzy logic, and to an extent, dynamic programming are utilized in this research to develop a code known as "UNCLE SCROOGE" (UNburdening through CLustering Effectively and Self-CROssover GEnetic algorithm). While initially examining the Traveling Salesman Problem, where an agent must visit each waypoint in a set once and then return home in the most efficient path, the work's end goal was a variant on this problem that more closely resembled the issues a UAV swarm would encounter.

As an extension to Dr. Obenmeyer's "Polygon-Visiting Dubins Traveling Salesman Problem", the Multi-Depot Polygon-Visiting Dubins Multiple Traveling Salesman Problem consists of a set number of visibility areas, or polygons that a number of UAV's, based in different or similar depot must visit. While this case is constant altitude and constant velocity, minimum turning radii are considered through the use of Dubins curves. UNCLE SCROOGE was found to be adaptable to the PVDTSP, where it competed well against the methods proposed

by Obenmeyer. Due to limited benchmarking ability, as these are newly formed problems, Obenmeyer's work served as the only basis for comparison for the PVDTSP. UNCLE SCROOGE brought a 9.8% increase in accuracy, and a run-time reduction of more than a factor of ten for a 20 polygonal case with strict turning requirements. This increase in performance came with a 99% certainty of receiving the best found solution over the course of 100 runs. With only a 1% chance for error in this particular case, the hybridized method has been shown to be quite powerful.

While no comparison is currently possible for MDPVDMTSP solutions, UNCLE SCROOGE was found to develop promising results. On average, it takes the code 25.62 seconds to approximately solve a 200 polygon, 4 depot, 5 UAV's per depot problem. This polygon count was increased even up to 2,500, with a solution taking 9.8 hours. It has been shown that UNCLE SCROOGE performs well in solving the MDPVDMTSP and has acceptable scalability.

.

ACKNOWLEDGEMENTS

Author would like to thank Dr. Kelly Cohen at the University of Cincinnati, as well as Dr. Corey Schumacher, Dr. Karl Obenmeyer, and Steve Rasmussen at Wright-Patterson AFRL for their expert advice and support of this research.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

UAV's are being utilized increasingly more often as the technology develops. The capability of complete autonomy will push the envelope of UAV applications even further. These autonomous systems could be utilized as backups during communication outages, or as the primary navigation operators of each UAV. Proper target allocation and path planning are necessary parts of this progress. A variant of the Traveling Salesman Problem (TSP), the Multi-Depot Polygon-Visiting Dubins Traveling Salesman Problem (MDPVDMTSP) is the subject of this thesis. This extension of Dr. Obenmeyer's PVDTSP [1] is a new problem that more closely resembles the problems encountered by UAV swarms.

This chapter will focus on the motivation, objective, problem statement, and assumptions of this research.

1.1 Motivation

An autonomous squadron of fully functional fighter-bombers would bring an increase to the safety of our servicemen and the capability to litter the skies with as many attack drones as our factories can output. However, there are many more tangible and realistic benefits of this work for the given year and technology that would be much less of a logistical, political, and ethical nightmare. A swarm of autonomous reconnaissance UAV's that can navigate an environment cooperatively, avoiding obstacles and needing a link to a human-operated computer only to display results has numerous uses.

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 [24, 25]. 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.

Firefighters, police officers, SWAT teams and other professions that deal with dangerous environments inside buildings or other environments where overhead fly-bys are not useful are prime examples. Outside of the world of UAV's, this knowledge could be useful to robotic manufacturing or surgery, and perhaps one day medical nanites that can cooperatively hunt cholesterol. While seemingly farfetched, current developments in many fields of science and engineering make the list of possible applications of this digital research seemingly endless.

1.2 Thesis Objective

The objective of this thesis is to offer an algorithm for large-scale multi-agent resource allocation, visibility, and flight-path planning problems encountered by a swarm of UAV's that provides near-optimal solutions for the constant velocity, constant altitude, and two-dimensional case.

In this research, multiple scenarios are studied, each containing an added complexity to the last. Being aimed towards use with UAV swarms, the first situation analyzed is the traditional Euclidean Traveling Salesman Problem (TSP). Here a starting depot is provided from which an UAV must depart, visit a set number of static points in the X,Y plane, and then return. Aircraft dynamics are ignored in this case; the path is simply a collection of straight edges between waypoints.

Additional nuances are added to the scenario until the problem now consists of multiple UAV's, flying at a constant altitude and velocity, based out of a single or multiple bases, that must encounter at least one point of all visibility polygons in a set, return to their corresponding depot, and do so in the most efficient manner possible. Thus this work is aimed to:

1. Simulate a two-dimensional, multi-depot, multi-agent, polygon visibility resource allocation and path planning problem
2. Develop a novel approach to approximate the MDPVDMTSP
3. Benchmark Matlab results to other methods, when possible
4. Determine the usefulness and limits of this methodology

1.3 Problem Statement

The problem of creating UAV swarm target allocation and path planning algorithms is indeed a complicated one. This can be approached through different means; however this study begins by examining the Traveling Salesman Problem (TSP). The Travelling Salesman Problem is a NP-hard (non-deterministic polynomial-time hard) mathematical and computational scenario that has a wide array of applications as described by Korte and Vygen [2].

In the most basic form, the TSP involves finding the shortest path, referred to as tour, of visiting a given set of cities and returning to the starting point. A city can be any necessary target, with distance of round trip being a measure of fitness, whether the actual variable be cost, time, etc. The TSP has seen applications in several diverse areas such as aerospace, logistics, genetics, manufacturing, telecommunications, and neuroscience [3].

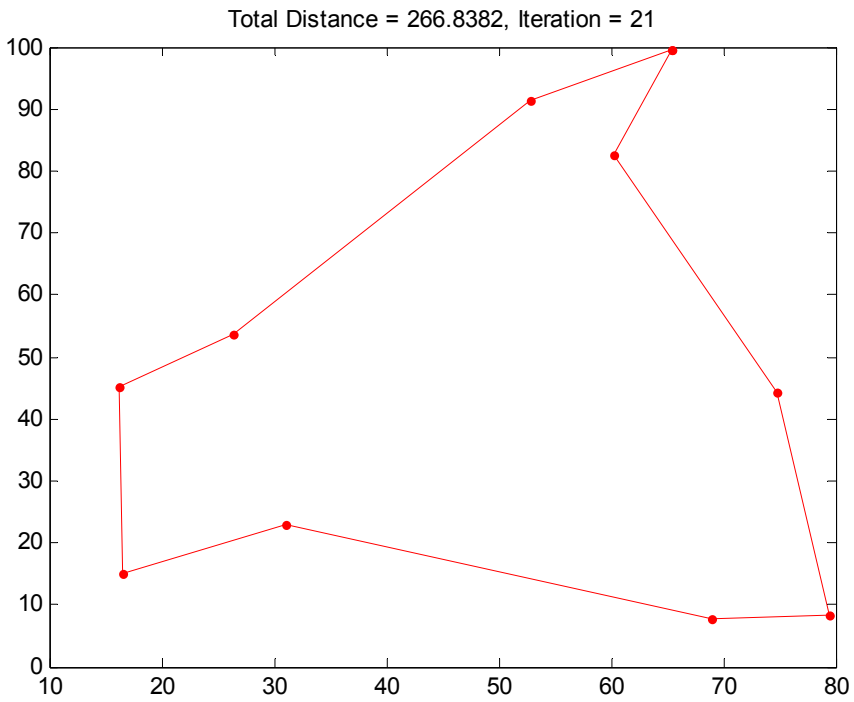

Figure 1: Example of a 10 City TSP Solution

The TSP lends itself to formulate the assignment of a UAV to multiple tasks as depicted by Rasmussen and Shima [4] and Goldberg [7]. In recent times, research has focused on finding new and unique strategies which enable UAV teams to optimize the use of their combined resources to accomplish their mission given the need for real-time task allocation using learning methods based on artificial neural networks [5,6].

In order to incorporate a UAV swarm, the problem evolves to the Multiple Traveling Salesman Problem (MTSP) and the Multi-Depot Multiple Traveling Salesman Problem (MDMTSP). These problems are intuitive, simply switching the objective to finding the most efficient path for a set of UAV's to visit each target, where each target can come out of either the same or different bases or depots as a starting and returning location. This problem has been examined in varied scales as a representation of cooperating UAV's, as by Shima et al. [8]

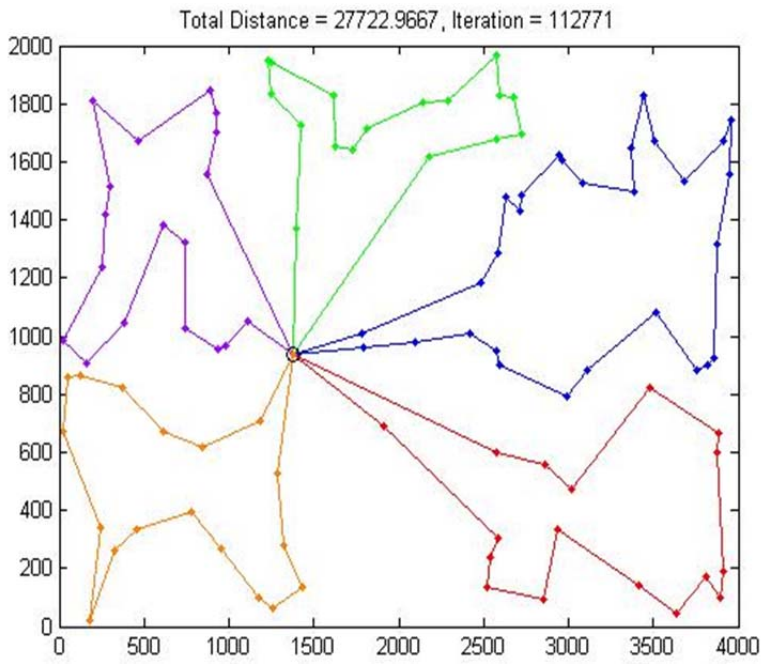

Figure 2: Example of a 100 city MTSP solution

As an extension of Dr. Karl Obenmeyer's work while at Wright-Patterson's AFRL [reference], the Polygon-Visiting Dubins Traveling Salesman Problem (PVDTSP) includes other complications. Here the point targets are switched to a polygon of any size in which the UAV must pass through at some point. This is a reduction of a visibility scenario, in which a sphere of some radius represents the effective camera or weapon range of the UAV. Buildings, mountains, and other such cover could block a portion of this hemisphere that is above ground. Taking a slice of the remaining hemisphere, assuming constant altitude, creates this visibility polygon. An illustration by Obenmeyer depicts this clearly:

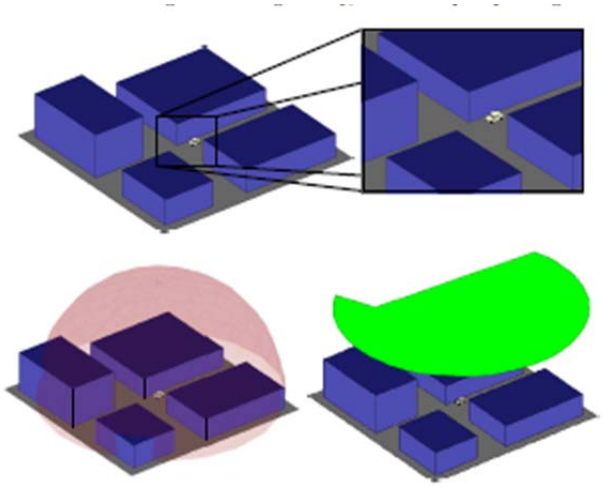

Figure 3: Breakdown of the creation of visibility polygons [1]

If aircraft dynamics are included, in a most basic form, some type of constraint on turning is required. Utilizing Dubins curves [9] a constant velocity is set, and as with most vehicles, a set minimum turning radius coincides with this constant velocity. Utilizing this view of simplistic vehicle dynamics, the most efficient path between two poses (coordinate and heading) is a combination of minimum turning radius turns and straight paths. Combining all of these forms the Multi-Depot Polygon Visiting Dubins Multiple Traveling Salesman Problem (MDPVDMTSP), which is the problem this thesis attempts to solve.

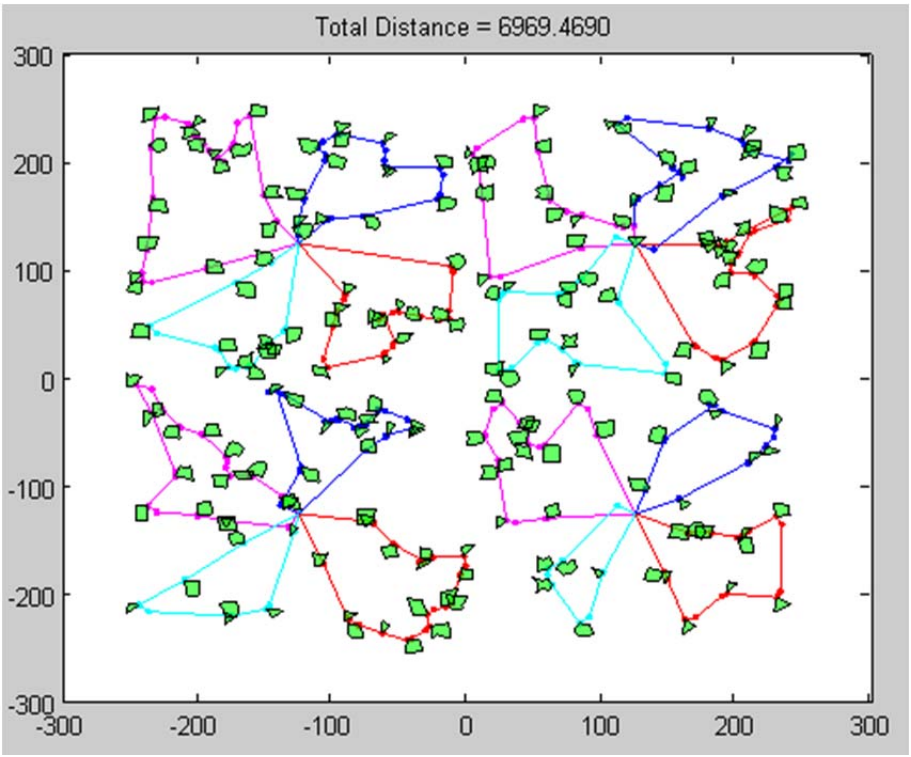

Figure 4: Example 4 depot, 4uav/depot, 200 polygon MDPVDMTSP case

With these added constraints, the problem becomes much closer to a realistic scenario than the base TSP, and acts as a proper proving ground for these methods in attempt to produce a fully-functional software.

1.4 Assumptions

Some simplifying assumptions are necessary to begin examining such a complex problem in this method. The Euclidean Traveling Salesman Problem in two dimensions assumes constant height and velocity, and neglects any aircraft dynamics (turning radii). The methods utilized (Chapter 3) certainly can be applied to three dimensions relatively simply. However, for the sake of the problem requirements, only a two-dimensional case is examined in this study.

Each base or depot is assumed to have unlimited simultaneous take-off and landing capabilities. That is, an infinite number of UAV's are allowed to return to the base at the same time without need of loitering. For the polygon-visiting Dubins case, a minimum turning radius is implemented, assuming constant velocity. While this does not incorporate aircraft dynamics, this simplification serves as a good starting point to vie away from the unrealistic application of the base TSP to UAV path planning.

For the Euclidean TSP, the distance from one point to another is the same in the opposite direction. This implies that for the method in this study, wind is neglected. The total number of waypoints is constant, and obstacles are not present. Lastly, the waypoints, in both the point and polygonal cases, are static.

CHAPTER 2: LITERATURE REVIEW

A review of the literature related to the research will be discussed here. As the methods presented by Dr. Obenmeyer [1] are the only current comparison possibilities for the PVDTSP, his work will be a primary focus. No publically available work could be found for benchmarking the proposed algorithm's performance in solving the MDPVDMTSP, or similar problems. The techniques utilized by the research will also be discussed.

2.1 Roadmap Methods

The sampling-based roadmap method utilized by Obenmeyer [1] was created through a multi-step process to approximate the PVDTSP. The complexities of which will not be fully reviewed in this study.

The construction of this method begins by sampling a finite discrete set of poses, which consist of an x, y, and heading value. The PVDTSP-feasible set of Dubins paths consist of all possible Dubins paths, with a given minimum turning radius, that can solve the proposed problem. The Finite One-in-a-set TSP (FOTSP) is where the vertices are now "finitely many nonempty mutually exclusive vertex sets called clusters" [1]. The FOTSP's solution is determining which vertex is to be selected for each polygon.

The poses in each polygon are sampled using a Halton Sequence, a quasirandom sequence. The use of a Noon-Bean Transformation allows an approximate conversion of a FOTSP to an asymmetric TSP. The ATSP is simply a TSP with directional costs, to which many solvers currently exist. The solution to this ATSP instance thus contains the solution to the PVDTSP. The method is referred to as "resolute complete" since it "provably converges to at least as good as any nonisolated solution as the number of samples in the roadmap increases" [1].

2.2 Fuzzy Logic

Developed by Lofti Zadeh in 1965 [10], fuzzy logic proposes a different way to describe information about continuous variables. Becoming increasingly popular over the past few decades, many published products that utilize Fuzzy Logic Systems (FLS's) have been quite successful. This approach allows an input to be included in multiple classification groups instead of a singular one, for example an object can be classified as some combination of blue and yellow, rather than green, or running at some percent power, rather than on or off. This type of classification, known as fuzzification, allows us apply a combination of if then rules based on membership in these groups. This rule base, developed from expert knowledge of the system, can solve many problems without investigation into complex equations and other

computationally expensive (or even impossible) obstacles. Fuzzy logic is discussed in detail in many texts and papers. As such, only a brief overview will be presented here.

Fuzzification allows the computer to handle information in a method more similar to our brains. Utilizing this process, a crisp value is given, and input into a membership function. This function maps out to what degree this crisp input belongs to the group, ranging from 0 to 1, rather than 0 or 1. There are usually a variety of membership functions in a given Fuzzy Logic System (FLS) that a crisp input can belong to over its domain. The breakdown of this crisp input into its level of membership in these functions is fuzzification.

In the example shown below, 3 membership functions exist, covering input values ranging from 0 to 10. These membership functions are triangular, but trapezoidal membership functions operate similarly. This particular scenario could represent load on a circuit or pressure in a tank, with the three membership functions being "Low", "Medium", and "High". The crisp input of 3 belongs to multiple membership functions.

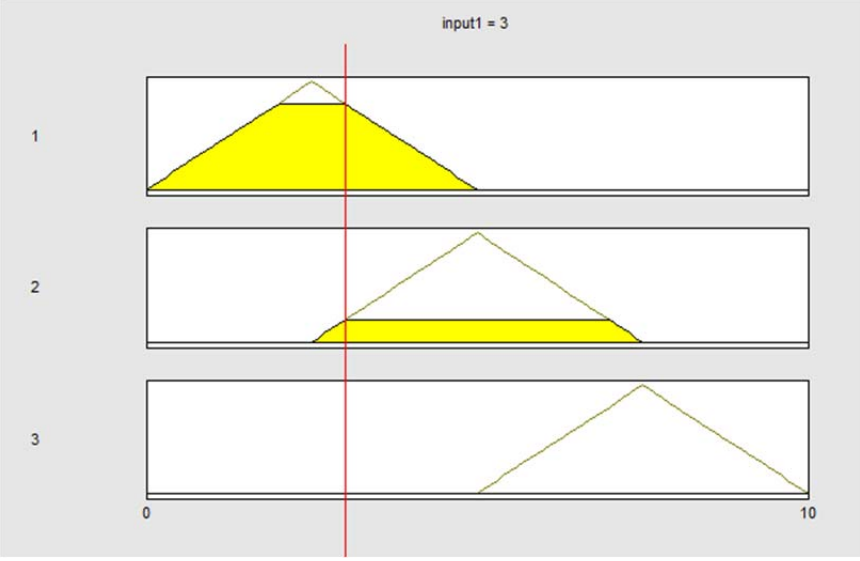

Figure 5: Example membership functions and fuzzification

As seen in Figure (5), this input belongs mostly to membership function 1. Based on its percent membership in each function, a rule base determines the crisp output of the system. This rule base is dependent on both expert knowledge and common sense to form a series of "if then" rules.

Our brains handle a vast multitude of complex problems on a daily basis. These problems often have incomplete or uncertain information that may not be easily quantifiable by a single crisp value, such as "on" or "off". This fuzzy information is usually not considered by machines which see only in binary numbers. Thus machines cannot typically utilize these rules that we live by, such as if a pot is "too hot to touch, remove your hand." Instead a machine would operate with a line of script along the lines of "if temperature exceed 110.0 degrees, remove hand". For such examples these lines of logic can typically handle most needs, however what if a less quantifiable input is examined? When we decide how much to tip at a restaurant, we base the decision typically on how good the food tasted, the quality of service provided, and perhaps the cleanliness of the building. Defining an exact measurement for what rude service or tasty food is presents a weakness in only examining crisp or specific data. Fuzzy logic allows the computer to operate with inputs such as "good" service, or "average" food.

Which membership function the rule base will see is variable, but a typical approach is the centroid method. The combined area of the membership in each function is created, as seen in the previous figure, and whatever membership function the centroid of this combined area lies in is what the rule base will activate off of.

Output membership functions are created similarly to the input functions. The final product of this process will take a crisp input such as vehicle speed, and through fuzzification

determine that it mostly lies within the membership function "way too high". The rule if "way too high", then "apply strong braking" will activate, and a centroid method can be employed again to determine the actual crisp output that the braking system will receive. This last process it termed "defuzzification".

This all allows Fuzzy Logic Systems to be utilized as a very strong approximation technique. While not an optimal technique for every situation, this can be very useful when any combination of un-quantifiable, partial, or uncertain information is all that is available [11]. Additionally, in a computational sense, when acquiring exact information is possible, but only through very expensive and lengthy calculations, it may be better to avoid this and work with what is available through such a system.

2.3 Genetic Algorithms

While evolution was investigated in computers by Barricelli in the 1950's, and further developments of the technique brought by Holland in the 1970's, genetic algorithms did not see much industrial application until the late 1980's [7]. Since then, this approximating technique has been utilized in a multitude of applications due to its very lax requirements and effective capabilities. Based on the theories of evolution and natural selection, genetic algorithms (GA's) have become a shining example of effectively implementing a characteristic of nature to solving complex mathematical and engineering-related problems [12].

Given a cost function, and a set of constraints dictating the necessary qualities of a solution, a genetic algorithm can be created. In this iterating computational process, a random population of possible solutions is first formed. Each of these random solutions must adhere to all constraints, and be evaluable by the given cost function. In GA terms, each solution is

referred to as a "string". Depending on the type of problem, each string can be a binary number, an array of integers, or a list of rules. Genetic programming [13] takes this a step further where each string is a length of code. As an example, if we wanted to minimize the function:

$$y = x^2$$

Over the range of:

$$0 \leq x \leq 31$$

Our strings would consist of binary numbers (combination of 0's and 1's) that are 5 digits long, as the binary number 11111 is equivalent to 31. Thus every possible string must be an array with length 5, with any assortment of 1's and/or 0's. Each string is then run through the cost function in order to determine its measure of optimality, commonly referred to as its "fitness". Since this is a minimization problem, a lower fitness measure is considered to be better.

A key quality of each string to be aware of are the "schema" that it contains. A schema is simply a subset of the string. The string:

1 1 0 1 0

Contains the following schemas:

* 1 0 * * 0

1 * * * *

* * 0 1 0

Where the asterisk (*) denotes a "don't care" position in the string. Patterns can easily be noticed after examining a population of strings as to which schemas are associated with higher

fitness strings. The importance of this will be discussed in more detail later in this chapter and the next.

It is at this point that the exact specification of what constitutes a genetic algorithm becomes less specific. However every genetic algorithm takes this population of strings, whose fitness has been determined, and emulates a generation of a biological population. The strong, or fit, have a higher chance to survive and produce offspring for the next generation. This process continues until a satisfactory solution is found. Such emulation can be done in a vast multitude of ways, but every method contains some form of polling the population. Most common methods include simply selecting the top X % of the population, roulette wheel style polling, or tournament style polling [1].

In roulette wheel polling, each string is given a set percentage chance to be selected based on its fitness, with the sum of all chances totaling 100%. While every string can be selected, the stronger members of the population have a higher probability. Tournament polling shares this quality, but through a different approach. Here a tournament or polling size is predetermined. A group of this size (typically much less than the total population size) is selected at random from the entire population. The fittest member of this smaller group "wins the tournament" and is selected. Both of these methods are repeated multiple times for each generation.

The chosen strings then go through some form of breeding process. The exact mechanisms here vary greatly from one algorithm to the next, but often some form of copying, crossover, and mutation is implemented. Copying refers to the chances that the chosen string will simply be moved into the next generation's population without any modification. Through crossover, two selected strings are paired, and a crossover point selected. The strings swap their

members, or "genetic material" with each other over this crossover point. For example, given two strings:

1 1 | 0 1 0

and

0 0 | 1 1 1

Two different strings would be produced as offspring and placed into the next generation's population. The resultant strings would be:

1 1 | 1 1 1

and

0 0 | 0 1 0

Crossover often results in drastic recoding of the strings' structure. As seen above, many possible schemas are disturbed by crossover, as more than half of each string is altered in this case. Crossover has some key strengths compared to other mechanisms; its large changes to the string help the initial population quickly become filled with more reasonable solutions and provide assistance in mitigating (not avoiding) local minima. Not every breeding mechanism is this drastic.

Mutation is another common breeding mechanism; however this acts on an individual string rather than a selected pair. This mechanism alters smaller sections of the string, rather than large sweeping portions. Similar to mutation in DNA, mutation can be the swapping of a specific 0 to a 1, replacing the 2^{nd} digit in the string with the 4^{th}, or any other desired operation. The key point here is that smaller changes are made to the string. This alters fewer schemas, and is capable of making a good string better with less chances of disturbing it too much.

Other mechanisms are popular, and many algorithms employ a variety of them. With some creativity, countless are possible. The decision of which mechanisms to utilize should be based upon the complexity of the algorithm. Genetic algorithms are very susceptible to damaging local optima; a population of strings can easily begin to settle towards one and may have great difficulties in moving past it. For simple problems such as the function in this example, there are no local optima, only the global minimum at $x = 0$. Thus a set of quick and dirty mechanisms should be created. For more complex problems, such as the TSP, local minima are very abundant. This type of problem demands a variety of mechanisms, some aimed toward rapid improvement of fitness, others more focused on mitigating and moving past local minima.

How the populations alter from one generation to the next must also be decided. Some algorithms keep a set percentage of the old population and only replace a portion of it to create a new equal sized generation, and others create an entirely new population each time [14]. These iterations of generations can be terminated in a number of ways. If an optimal bound of the cost function is known, a string having such a fitness value could end the process. The algorithm can be told to run through a set number of generations, or coded to end when the optimal value has not improved after a set number of generations.

While investigating genetic algorithms, one would do well to also examine the processes undergone to domesticate certain animals and select breeds. While some important factors differentiate a genetic algorithm's optimal process from that of the recent domestication of the silver fox experiments in Russia [15] for example, there is knowledge to glean from these studies. Each foxes traits are comparable to the schema in a string, but the genetic algorithm has the benefit of being able to run through multiple generations of populations in the same second.

With this capability, there is less of a need for "immediate results" as we would see in the regular domestication of an animal.

A good example of the downsides of this rush is the continual genetic defects found in dogs that are bred for pedigree and certain traits, such as size or color. Common issues in certain breeds include inclination towards hip problems, blindness, ear infections, and even cancer. It has been shown that mutts, or cross-breeds, have less inclination to share these issues with their purebred counterparts, and are much more likely to live healthier lives [16]. What the genetic algorithm designer should take away from this is that creating a setup that is extremely biased towards only the fittest members in the population has the potential to have a particular string or set of schema overpower a population. While this will lead to a quick improvement during the beginning iterations, it will quickly lead to damaging local optima that the algorithm will struggle to surpass. By maintaining some level of diversity in the string population, the initial progress will be slower, but this problem can be avoided to a degree. This is accomplished by ensuring that the bias to select and benefit the fittest strings is not overbearing.

2.4 Discussion

Found within sections 2.2 and 2.3 are the methods that form the basis of this research. Genetic algorithms and fuzzy logic pair well together [17] and have been shown to hybridize to amazing results. The typical genetic-fuzzy system is a fuzzy system whose parameters are determined by a genetic algorithm. The genetic-fuzzy system outlined in this work does not follow this pattern.

Presented next in the thesis will be the development of a system that utilized genetic algorithms, fuzzy logic, and to an extent, dynamic programming in an effort to approximate the MDPVDMTSP. These control techniques work cooperatively, rather than layered inside one

another, in an effort to solve this very complex problem. This approach is feasible due to each method having separate and unique strengths.

CHAPTER 3: METHODOLOGY

The advantages of both genetic algorithms and fuzzy logic discussed in the previous chapter are why they are the foundation of this research. Here the actual construction of the UNCLE SCROOGE (UNburdening through CLustering Efficiently and Self-CROssover Optimized Genetic Algorithm) program will be described in detail.

3.1 Genetic Algorithm – "SCROOGE"

As mentioned before, a genetic algorithm needs to be customized to best fit the problem to which it is being applied. The following sections will break down the construction of each component of SCROOGE. Two prior publications [19, 20] display portions of this work.

3.1.1 String Structure

Each string of a genetic algorithm must adhere to the constraints of the problem. For the TSP, a solution is an array of cities, representing the order in which they are visited. All points must be visited, no point shall be visited more than once, and the UAV must return to the initial point at the end of the tour. While these constraints noticeably limit a genetic algorithm approach, this does not enforce a single string structure. Examining the following 5-city TSP, with cost function "J" will show this to be true:

$$J = \sum_{i=1}^{n} \sqrt{((y_{i+1} - y_i)^2 + (x_{i+1} - x_i)^2)}$$

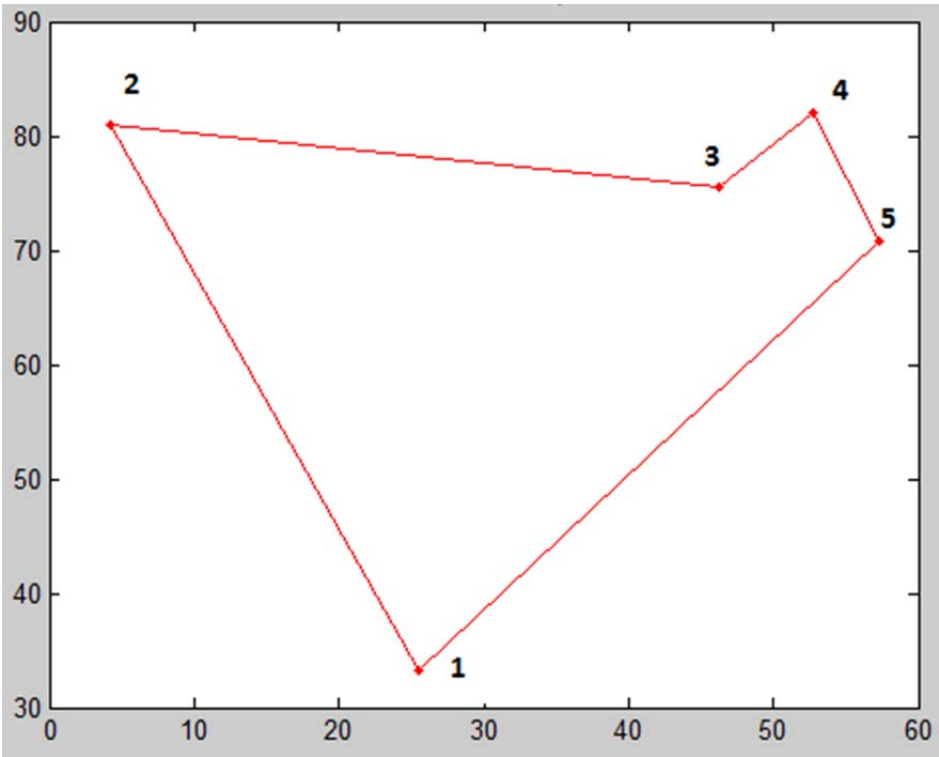

Figure 6: Example 5 city TSP

The optimal solution to this problem is quite simple to find, and it is clear that the solution shown above is indeed the optimal. An obvious way to depict this route as a genetic algorithm string is:

1 2 3 4 5 1

However, since the "1" at the end of the string is merely a copy of the initial member, we can remove this from the actual string and simply script the return to the initial member as a part of the cost function, resulting in:

1 2 3 4 5

Changing the cost function to:

$$J = \sqrt{((y_1 - y_n)^2 + (x_1 - x_n)^2)} + \sum_{i=1}^{n-1} \sqrt{((y_{i+1} - y_i)^2 + (x_{i+1} - x_i)^2)}$$

A choice is present now to either lock the starting point as a constant, or allow it to change with the algorithm. If we do incorporate point "1" as a permanent starting location, the string size reduces again, thus making the genetic algorithm slightly less computationally expensive. However, it reduces the number of optimal strings to two, namely:

2 3 4 5

5 4 3 2

By doing this, the number of ways the optimal solution can be represented has been greatly reduced. If this were reduced to only one possible string representation, a strong benefit would be present in the fact that certain schema would be designated as always optimal. In reality however, it is clear that:

* 3 4 *

is a schema that does not always lead to optimality. If we do not lock the solution to a starting point, our string length is increased by one, but the number of possible representations is greatly increased, with a few examples shown here:

2 3 4 5 1

2 1 5 4 3

5 1 2 3 4

5 4 3 2 1

While the difference in string size may be noticeable for this small-scale example, the actual difference in computational cost for larger cases with points (n) greater than 50 is practically negligible. The presence of harmful local minima in the 5-city case is relatively low, but for a scenario such as in the following figure, they are much more pronounced.

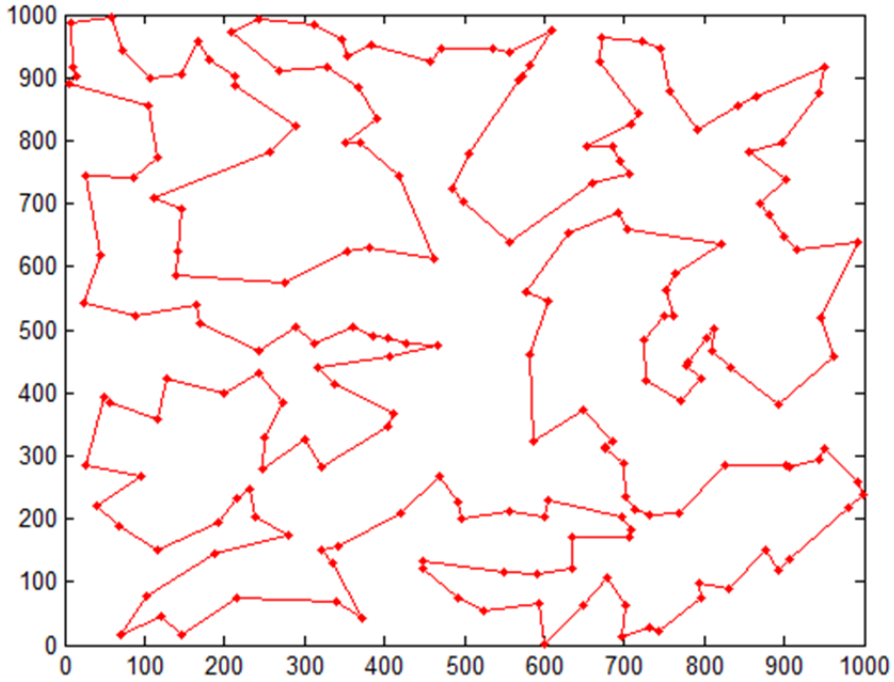

Figure 7: Example 200 city TSP

With this in mind, SCROOGE was structured to not have the UAV's depot be locked as the initial member in the strings. Since it does not matter either way in the eyes of the cost function, and it provides more possible string representations of the optimal solution, it is preferable in this case.

3.1.2 Polling Style and Generation Structure

Since local optima abound in the TSP, the polling style and generation structure must be chosen with this in mind. After studies of a variety of methods, it was noted that the tournament

style polling was a more powerful tool than roulette wheel polling, with different tournament sizes seeming to be effective at different iterations into the simulation. During the onset of the simulation, a large tournament size enabled the population to fill with reasonable solutions quickly, without too much bias to a particular string. As the run progressed, the optimal tournament size lowered, and was reduced even further towards the end of the run. This is logical, as the need to mitigate local minima increases with iteration, and the smaller tournament size allows the weaker strings to have a chance to breed and perhaps introduce helpful new schema.

Since tournament polling selects a random group of strings, with no bias towards fitness, to compete with each other, the odds of a weak group was often noted to be greater than he odds of the weaker strings being selected through roulette wheel polling. While this guarantees that the weakest string of a generation will never breed, it has a higher chance of promoting moderate diversity. Thus morphing parameters were introduced to SCROOGE, with 3 phases; Initial, Intermediate, and Final.

In an effort to further this goal, a new population is created every generation. That is, for a population size of n, n groups of randomly selected strings are created. The winner of each goes through the reproductive process to create 1 string for the new generation. Thus the next generation maintains the size of n.

3.1.3 Breeding Mechanisms

With the selected string structure, and with most string representations of the TSP [1], basic crossover is unable to meet the constraints of the problem.

Parents	Crossover
1 2 3 4 5 6	1 2 3 : 3 2 1
6 5 4 3 2 1	

Table 1: Example of improper crossover for TSP

As depicted in Table (1), a child of crossover could easily contain duplicate point visits. There are methods to alter the crossover procedure that avoid this, shown by Goldberg [1]. Such approaches include partially matched crossover, order crossover, and cycle crossover. Kundu and Pal [18] propose a different sort of approach; self-crossover. This method is similar to asexual reproduction that organisms such as starfish can undergo. Here, one string crosses with itself in a set number of locations. An example being:

9 10 | 2 5 6 3 1 | 4 8 7

Would become:

4 8 7 | 9 10 | 2 5 6 3 1

While being a one-string mechanism, the effect of schema redistribution that normal crossover has is witnessed here as well. More self-crossover sites are utilized in larger strings to produce similar distributions. The fact that this method, which is in reality an extreme mutation, produces similar results to other forms of crossover is not its main strength however. Methods such as order crossover, or partially matched crossover, require more computational cost to execute due to their larger number of steps required. Despite the fact that these other mechanisms produce two offspring, the larger amount of coding required offsets this. Self-crossover has been found to be incredibly computationally efficient through this research.

Following a more traditional route, mutation is utilized in combination with self-crossover inside SCROOGE. Mutation's strength of small changes inside strings is focused on, and a good mutation is often found to be necessary to find the optimal solution of larger tours. When a string is chosen for reproduction it is checked against a crossover rate and a mutation rate, to see if neither, either, or both of these mechanisms occur. This percentage chance changes with time into the run, similar to polling size. Crossover rate is higher towards the beginning of the run, and lowers slightly towards the end. Mutation rate is relatively low at the onset of the simulation, but increases drastically over time. These changes are discrete rather than continuous; however more research into this area could perhaps develop a continuous distribution of these parameters. This will be discussed further in Chapter 4.

Self-Crossover and mutation combine to manipulate strings in manner shown below:

Parents	Self-Crossover	Mutation	Self-Crossover + Mutation
1 2 3 4 5 6	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 2: Possible breeding mechanisms within SCROOGE

3.1.4 Additional Mechanisms

SCROOGE contains a few extra parameters with each serving a specialized purpose during specific cases. During longer cases where finding the exact optimal is desired, SCROOGE can be set to not allow duplicates of any string to exist. It accomplishes this by simply changing the string to a different representation of the same solution, as shown in Chapter 3.1.1. This can help to mitigate local minima, but does slow performance.

Additionally, another phenomenon worth noting is the effective difference between breeding mechanisms based on string size. As discussed in Chapter 2.3, crossover typically has a more drastic effect on the string than mutation. However if one analyzes a 2 city TSP, with the following string:

1 2

Any type of mutation, or crossover on this string would result in the offspring:

2 1

While the difference in breeding mechanisms is more pronounced in a 3 city TSP, and so on, smaller case TSP's (less than 10 cities) can be solved as accurately by just one breeding mechanism. For such small cases, SCROOGE's is set to mutation only, providing a boost in computational speed without a loss in accuracy.

3.1.5 MTSP Implementation Issues

As presented in a prior publication [20] SCROOGE was successfully adapted to solve the Multiple TSP (MTSP). Since this research has eventual goals for a real-time program solving even more complicated problems than the TSP, the very lengthy run time of this adaptation (over 2 hours to solve a 100 city, 5 UAV MTSP). It was clear that the single TSP was the extent that SCROOGE should be pushed to, and that hybridizing with other heuristic methods was necessary to keep run-time at a minimum.

3.2 Fuzzy Logic – "UNCLE" System

In order to determine how to properly utilize a fuzzy logic system (FLS) to work alongside SCROOGE, the MTSP version of SCROOGE was observed during multiple cases. Below is Figure 2 again:

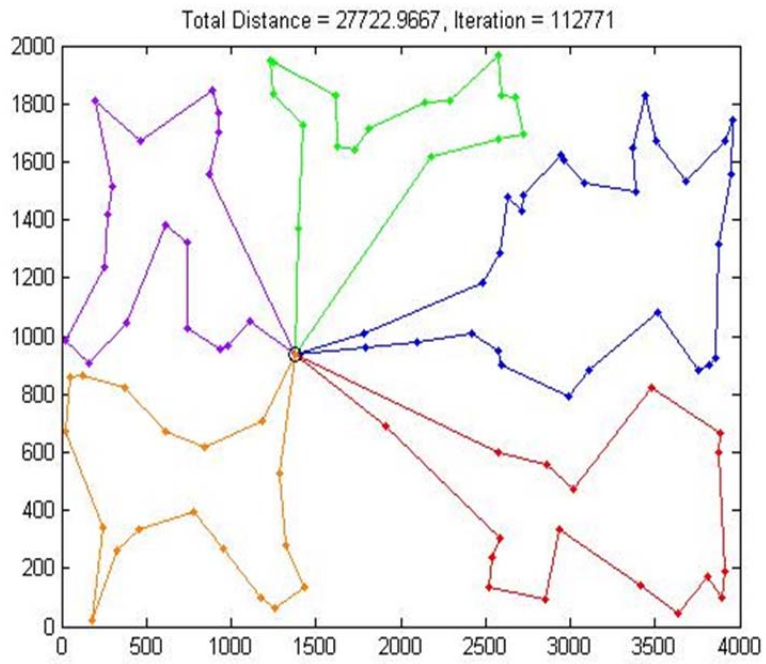

Figure 8: Figure 2 again, example 100 city, 5 UAV MTSP

Notice the constraints on the MTSP cost function; in this scenario each UAV has to go to a minimum number of points. Otherwise, the algorithm would output a solution where 4 of the 5 UAV's are inactive, and the 5th visits every city (as this would be the most efficient in terms of distance). The problem has now become a balance of optimizing total time and distance. Through the rest of this research, the multi-agent cases follow this rule. The algorithm can easily be changed to optimize purely for distance, or for time (a measure of the average distance of each UAV).

The figure shows each UAV receiving a cluster of cities in a mostly radial fashion. Keeping to this pattern, the UNCLE system was developed by first converting all of the points' coordinates to polar notation, with the depot considered the origin. Since the solver is outputting loop-like paths, the main focus of the polar coordinates is the angle rather than the radius.

Fuzzy c-means clustering, a variation on MacQueen's K-means clustering [21], takes a set of coordinate measures and forms a given number of clusters based on the minimization of the following cost function [22]:

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

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

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. This provides a drastic reduction in run-time, but slightly worse results.

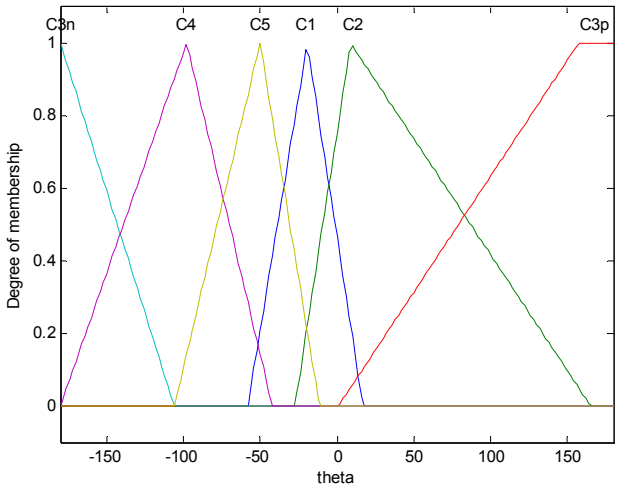

Figure 9: 5 UAV UNCLE input membership functions for random target distribution

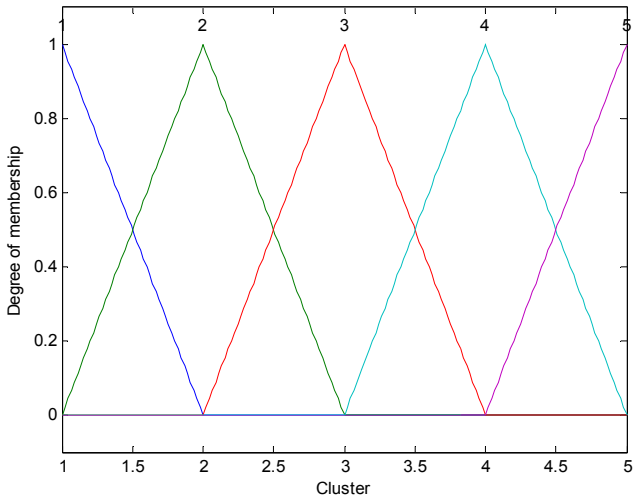

Figure 10: 5 UAV UNCLE output membership functions for random target distribution

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 [20]. 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 (9), 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.

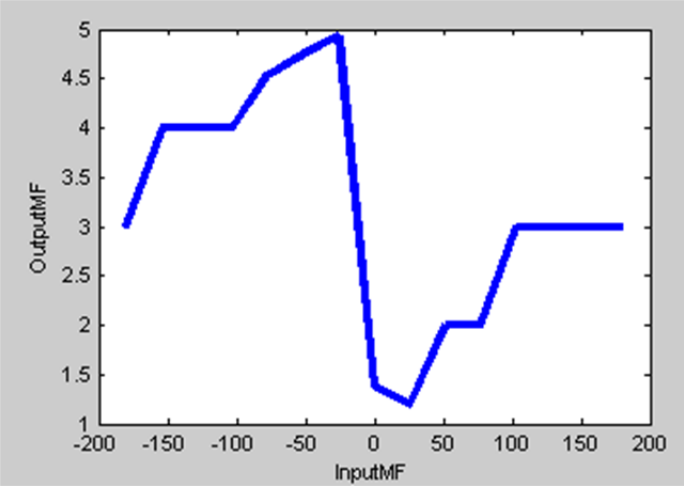

Figure 11: 5 UAV UNCLE rule surface for random target distribution

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 [20].

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 (9) 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.

While only a single input single output system, the UNCLE system prevents the scenario from being simply split by radial lines of set angles. By examining both TSP and MTSP solutions, and determining that optimal paths are loops where paths never intersect, this fuzzy logic system was determined to be an effective approximation. While further work must be done to perfect the system, the work thus far polished the clustering process nicely and produced effective results in a very quick manner. Including additional inputs may further the accuracy of UNCLE, and especially with the addition of complexities such as three-dimensionality, a multiple input system must be looked into.

It is important to note that UNCLE SCROOGE is an approximation technique. The MTSP has limited benchmarking capabilities, partially due to the variation in any given cost functions. The combination of all of the advanced control methods inside UNCLE SCROOGE is all that is necessary to begin solving such complex problems as the MDPVDMTSP. The process of developing this capability with UNCLE SCROOGE is shown in the proceeding chapter.

CHAPTER 4: DEVELOPMENT

Past publications [19, 20] have shown the effectiveness of both SCROOGE and the UNCLE system as a TSP approximator and clustering code. This chapter will focus on covering the development and optimization of the code and adapting UNCLE SCROOGE to the final aim of this thesis, the MDPVDMTSP. 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.

4.1 Adaptation to 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.

4.1.1 Multi-Depot MTSP

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.

4.1.2 TSP to Polygon-Visiting Dubins TSP

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:

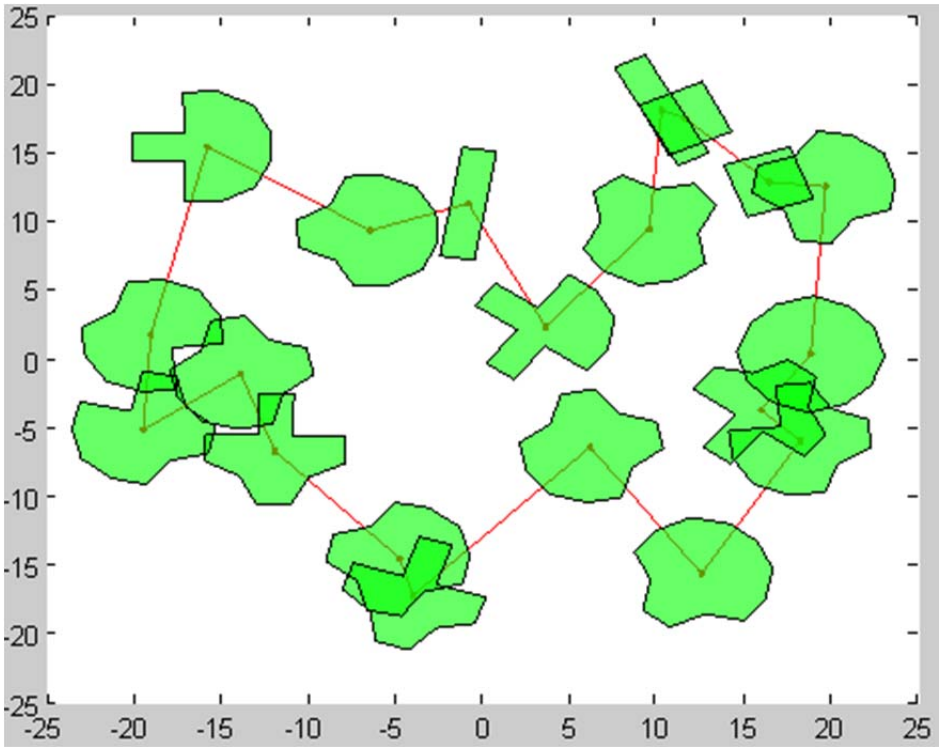

Figure 12: Initial phase of PVDTSP

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:

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

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

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

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

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

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:

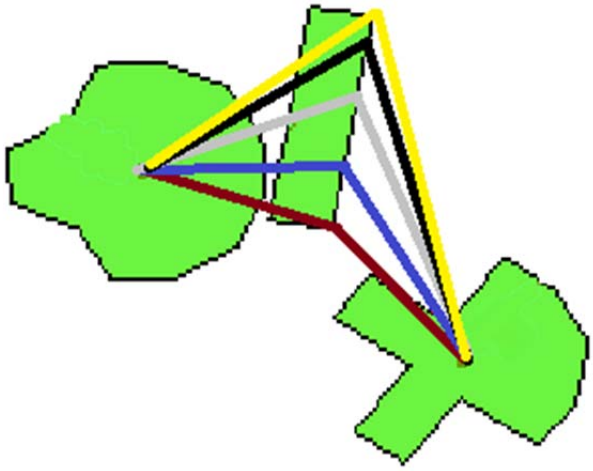

Figure 13: Polygon Point Selection Process

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 [23], shows some example poses, q, and their appropriate Dubins paths.

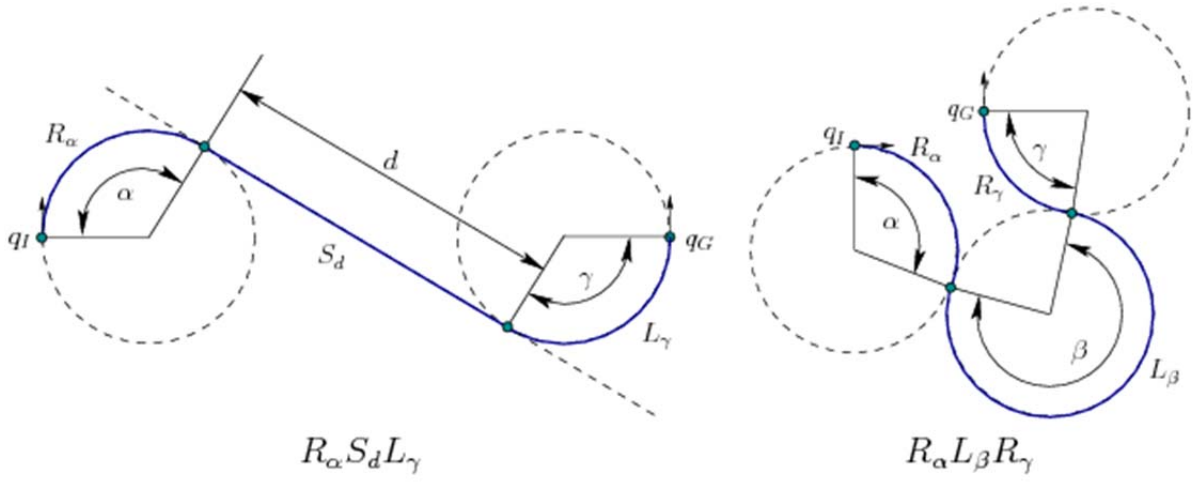

Figure 14: Example Dubins Paths [23]

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. The next chapter will focus on the results of this hybridized approach.

4.2 Optimization of UNCLE SCROOGE

During the infancy of this research, the code 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 [19]. 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 Chapter 4.1. 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 3: 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.

CHAPTER 5: RESULTS

The results of this research will be presented in this chapter. Since benchmarking is only currently possible for the PVDTSP, this will be shown first. Next, a variety of cases will be examined, showcasing the capabilities of the program. Lastly, some instances that depict the weaknesses of the code and possible areas of improvement are displayed.

5.1 Benchmarking of PVDTSP

As the first work investigating this particular variant of the TSP, Dr. Obenmeyer's research [1] 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.

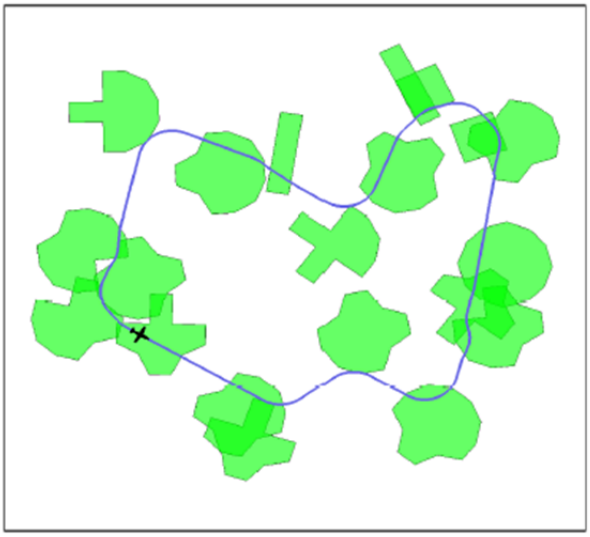

Figure 15: 20 Polygon PVDTSP [1]

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:

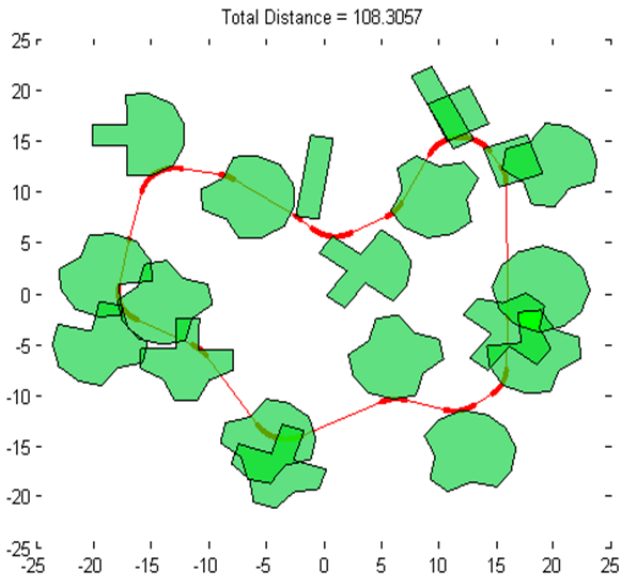

Figure 16: UNCLE SCROOGE's Solution to Sample PVDTSP

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 4: Comparison between "Resolute Complete" roadmap method and UNCLE SCROOGE

As seen in Table (4), 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++ [1]. 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:

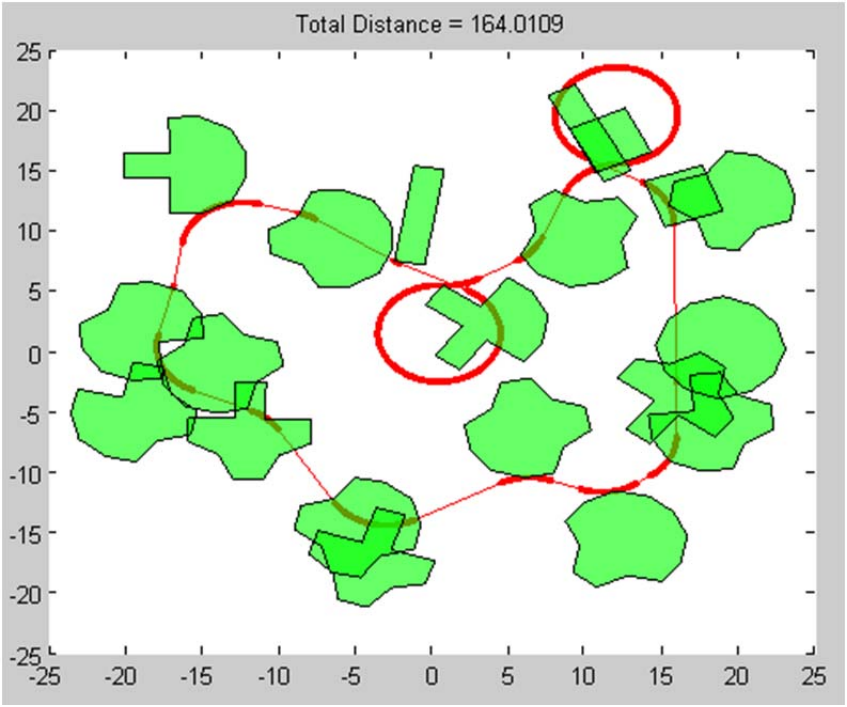

Figure 17: Increased Minimum Turning Radius Solution

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.

5.2 MDPVDMTSP Results

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 has been demonstrated in 5.1.

First to be presented is a relatively simple case, a 50 Polygon, 2 depot, 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.

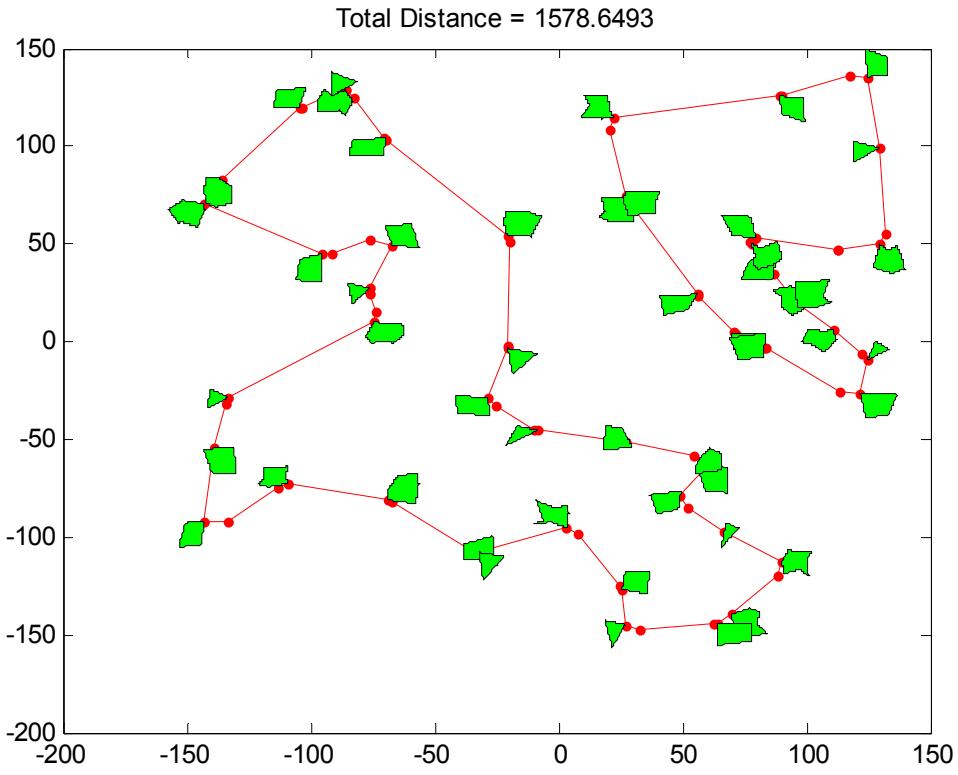

Figure 18: 100 Polygon, 2 Depot, 1 UAV/Depot Example 1

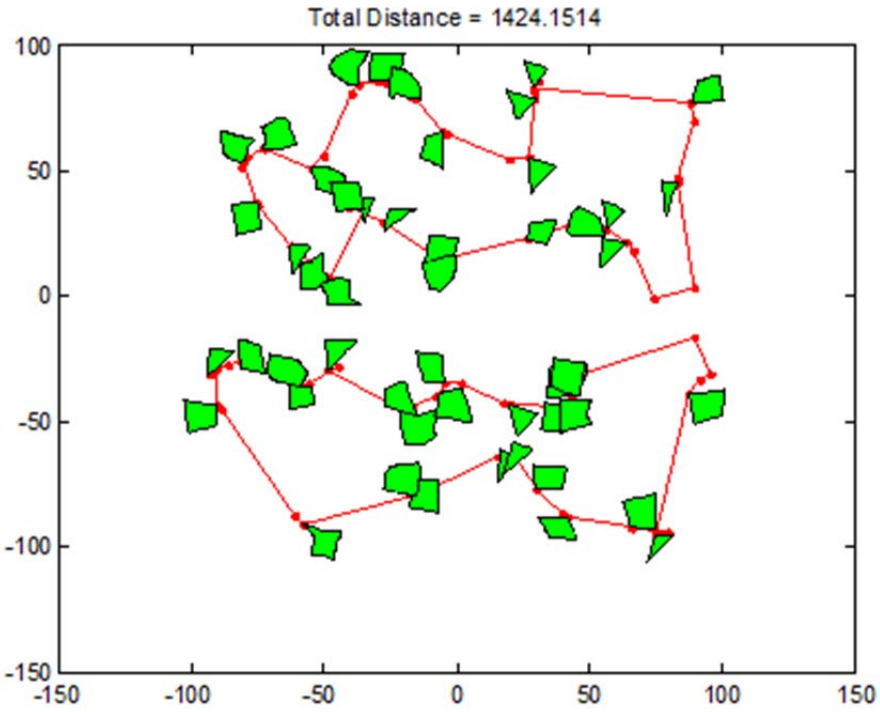

Figure 19: 100 Polygon, 2 Depot, 1 UAV/Depot Example 2

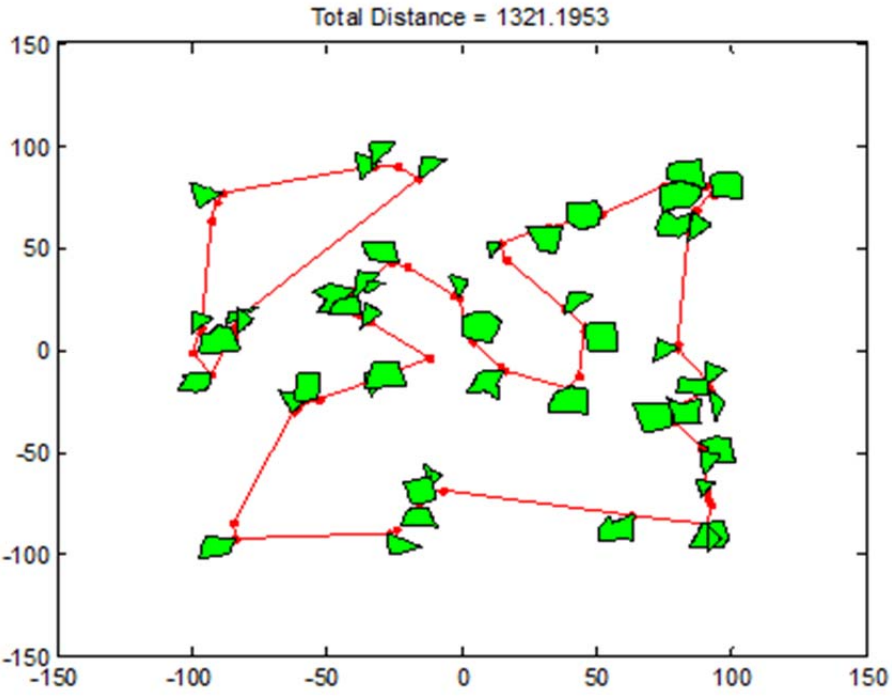

Figure 20: 100 Polygon, 2 Depot, 1 UAV/Depot Example 3

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 (18). 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:

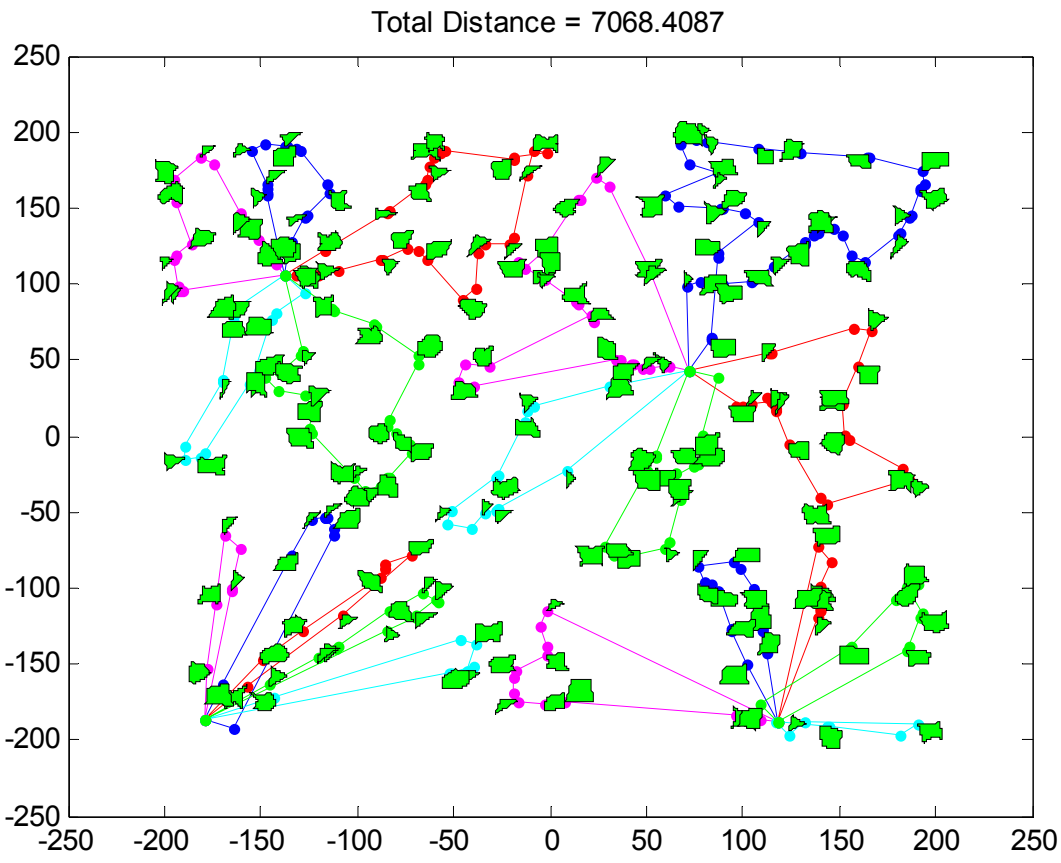

Figure 21: 250 Polygon, 4 Depot, 5 UAV/Depot Example 1

In the example shown in Figure (19), 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.

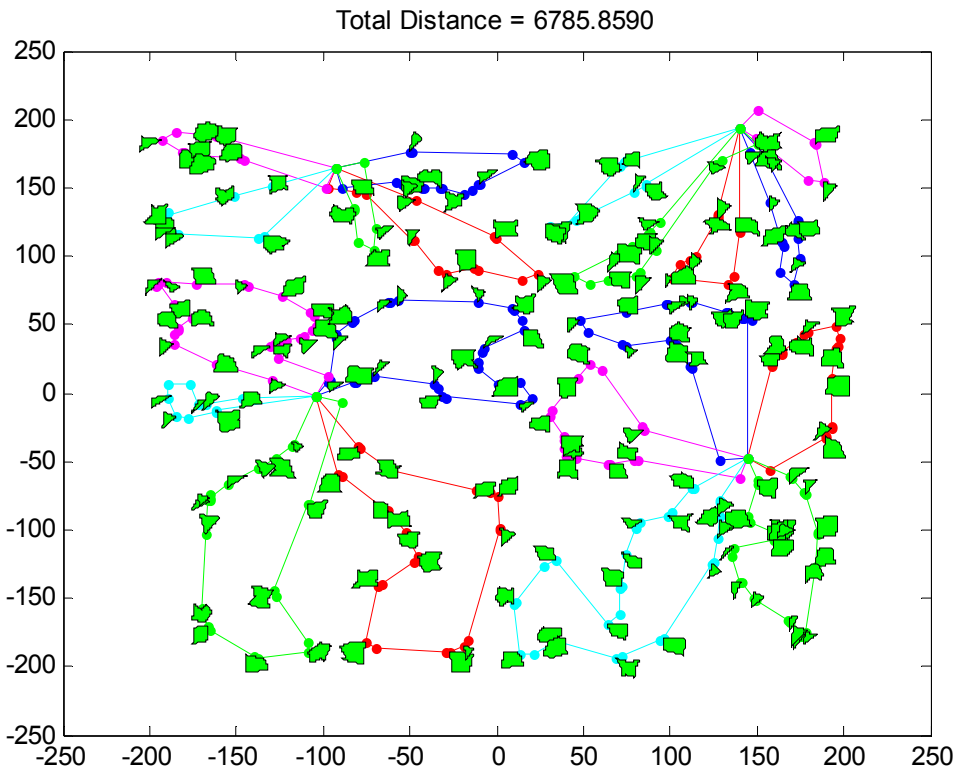

Figure 22: 250 Polygon, 4 Depot, 5 UAV/Depot Example 2

Figure (20) 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 [19], 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

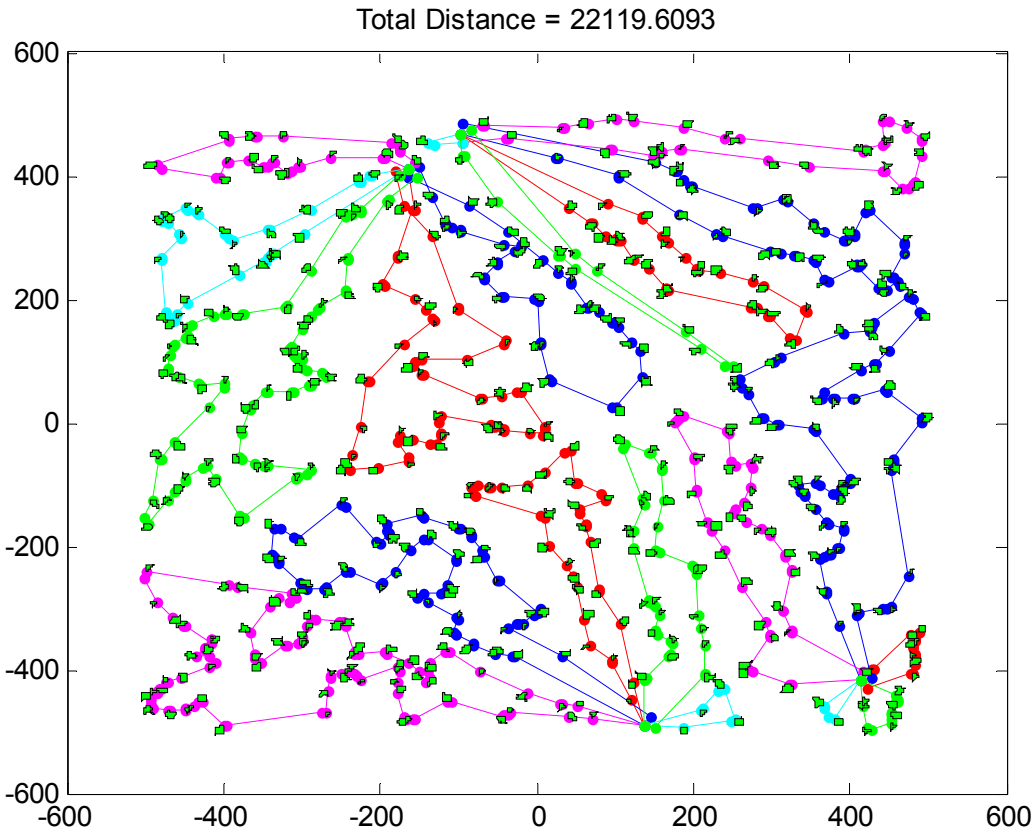

Figure 23: 500 Polygon, 4 Depot, 5 UAV/Depot Case

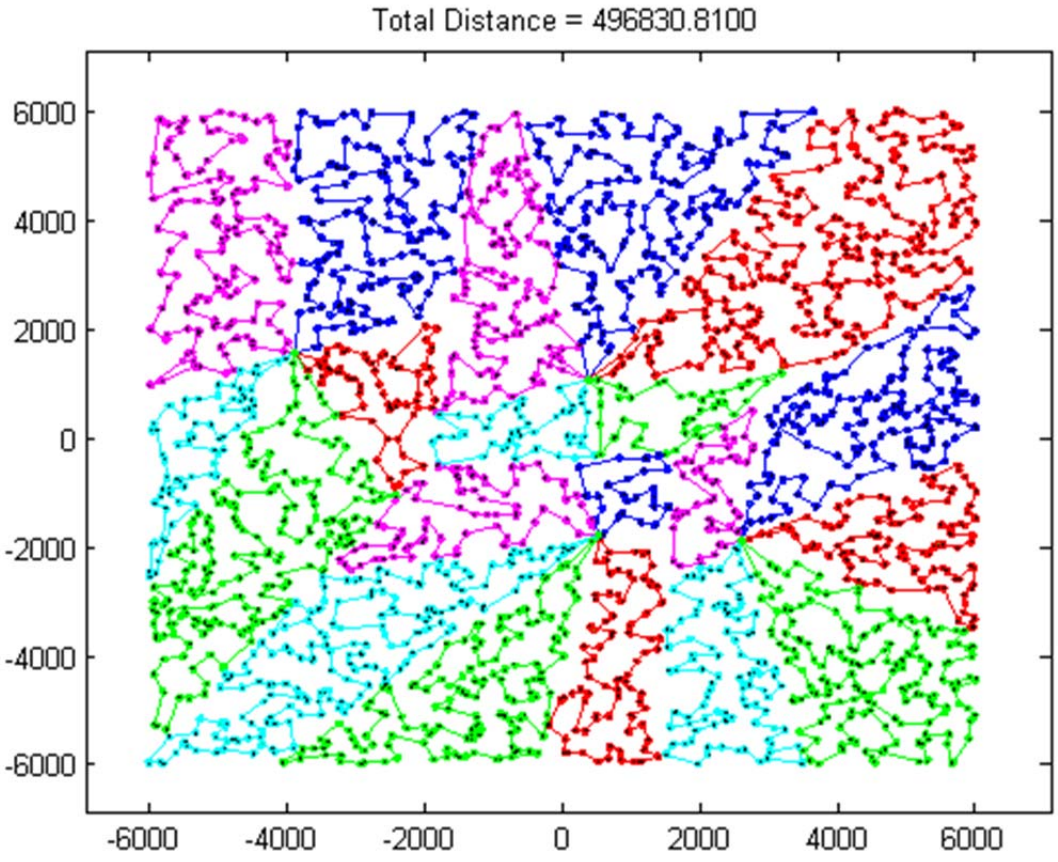

Figure 24: 2500 Polygon, 4 Depot, 5 UAV/Depot Case

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 (5) 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 5: Comprehensive Run-Times for Sample Cases

The impact of these results will be discussed in the following chapter.

CHAPTER 6: CONCLUSIONS

Displayed in this thesis is an approach to solving the Multi-Depot Polygon-Visiting Dubins Multiple Traveling Salesman Problem (MDPVDMTSP). 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 (5) 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.

CHAPTER 7: FUTURE WORK

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. The fruit of this research, UNCLE SCROOGE, has shown great capabilities in this area through a combination of powerful techniques.

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.

REFERENCES

1. 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.

2. Korte, B., and Vygen, J., 2008, Combinatorial Optimization – Theory and Application, 4th Edition, Algorithms and Combinatorics, Vol. 21, Springer-Verlag Berlin Heidelberg.

3. Applegate, D.L., Bixby, R.E., Chvatal, V., and Cook, W.J., 2006, The Traveling Salesman Problem – A Computational Study, Princeton Series in Applied Mathematics, Princeton University Press, New Jersey.

4. Rasmussen, S.J., and Shima, T., 2008, "Tree Search Algorithm for Assigning Cooperating UAVs to Multiple Tasks", International Journal of Robust and Nonlinear Control, Vol. 18, pp135-153.

5. Cohen, K., and Rasmussen, S., "Application of Proper Orthogonal Decomposition and Neural Networks to UAV Task Assignment", AIAA Infotech@Aerospace Conference, Atlanta, GA, April 20-22, 2010, AIAA Paper 2010-3541.

6. Humphrey, L., Cohen, K., and Rasmussen, S., "Application of Proper Orthogonal Decomposition and Artificial Neural Networks to Multiple UAV Task Assignment", Invited Paper, AIAA Guidance Navigation and Control Conference, August 2-5, 2010, Toronto, Canada, AIAA Paper 2010-8439.

7. Goldberg, D., E., Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Professional, 1st edition, 1989.

8. Shima, T., Rasmussen, S.J., Sparks, A.G., and Passino, K.M., 2005, “Multiple Task assignments for cooperating uninhabited aerial vehicles using genetic algorithms”, Computers & Operations Research 33 (2006) 3252-3269.

9. Dubins, L.E., “On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents”, American Journal of Mathematics, Vol. 79 No. 3 (July 1957), pp. 497-516.

10. Zadeh, L.A. (1965). "Fuzzy sets". *Information and Control* (3): pp. 338–353.

11. Sivanandam, S.N., Sumathi, S., Deepa, S.N., Introduction to Fuzzy Logic using Matlab, Springer-Verlag, 2007.

12. Ross, P., and Corne, D., “ Applications of Genetic Algorithms”, AISB Quarterly 89, Ed. T.C. Fogarty, 1994, pp. 23–30

13. McPhee, Nicholas Freitag; Poli, Riccardo; and Langdon, William B., "Field Guide to Genetic Programming" (2008). *Computer Science Faculty.* Paper 1.

14. Sallabi, O.M., and Younis, E.H., “An Improved Genetic Algorithm to Solve the Traveling Salesman Problem”, World Academy of Sciences, Engineering and Technology 52, 2009.

15. Kukekova, A.V., et al, “Measurements of Segregating Behaviors in Experimental Silver Fox Pedigrees”, *Behavior Genetics* 2007, pp. 185–94

16. McGreevy, P.D., and Nicholas, W.F., “Some Practical Solutions to Welfare Problems in Pedigree Dog Breeding”, Animal Welfare, 1999, Vol. 8, 329-331

17. O. Cordon, F. Herrera, F. Hoffmann and L. Magdalena, *Genetic Fuzzy Systems: Evolutionary tuning and learning of fuzzy knowledge bases*, Advances in Fuzzy Systems: Applications and Theory, Volume 19, World Scientific, 2001.

18. 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.

19. 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.

20. 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.

21. 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.

22. 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

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

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

25. Osborn, K., “Army surpasses 1 million unmanned flight hours”, 2010

http://www.army.mil/article/38236/army-surpasses-1-million-unmanned-flight-hours/