SPRAY TREATMENT PLANNING USING THE ANT SYSTEM METAHEURISTIC

by

RUCEN DENG

(Under the Direction of Walter D. Potter)

ABSTRACT

The spray treatment planning project is designed to assist the pesticide management programs as a computerized decision support system for optimizing spray routes. This optimization problem is simplified as the capacitated vehicle routing problem. Because the exhaustive search costs a great deal of time to solve the combinatorial optimization problems, Ant Metaheuristic is used to find the solution in a more efficient way. The algorithm of Ant Metaheuristic is originally developed based on the phenomenon of ants' foraging behavior which is a kind of indirect communication by leaving pheromones. The results from Ant Metaheuristic are compared with the results from the exhaustive search and the Genetic Algorithm. The software CASPER is found to be a useful tool for calculating the results for different routes. Although the Ant Metaheuristic result is not optimal like the exhaustive search, the result is considered to be good enough, taking into account the program runtime and efficiency.

INDEX WORDS: Heuristic Search, Ant System Metaheuristic, Capacitated Vehicle Routing Problem, Traveling Salesman Problem, Ant Colony Optimization

SPRAY TREATMENT PLANNING USING THE ANT SYSTEM METAHEURISTIC

by

RUCEN DENG

B.S., Huazhong University of Sci. and Tech., China, 2004

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2006

© 2006

Rucen Deng

All Rights Reserved

SPRAY TREATMENT PLANNING USING THE ANT SYSTEM METAHEURISTICS

by

RUCEN DENG

DEDICATION

This thesis is dedicated to my family for they are always there with encouragement and support.

ACKNOWLEDGEMENTS

I would like to thank Dr. Potter and all of the professors and friends for their support and guidance.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS v

LIST OF TABLES viii

LIST OF FIGURES ix

CHAPTER

1 Introduction 1

1.1 Heuristic Search 1

1.2 The Problem 3

1.2 Ant Metaheuristic ---Ant Colony Optimization(ACO) 4

2 Background 10

2.1 Problem Domain 10

2.2 Detailed Ant System Metaheuristic 14

3 Methodology 20

3.1 Approach 20

3.2 Experiment Setup 25

4 Results 27

5 Summary 29

5.1 Conclusions 29

5.2 Future Directions 31

REFERENCES 32

APPENDICES 36

A Flight Plan Result from ACO code 36

B Data File Used 38

C Part of Program in Phase 2 39

LIST OF TABLES

Page

Table 2.1: Parameter Setting for ACO Algorithms without Local Search 18

Table 3.1: Needed Pesticide Calculated by the Area Accounting for Exclusion.....22

Table 3.2: Needed Pesticide Calculated by the Area Ignoring Exclusions.....23

Table 4.1: Statistics Comparisons by Java Program.....28

Table 4.1: Statistics Comparisons by CASPER.....28

LIST OF FIGURES

Page

Figure 1.1: Ant are the Insects the ACO algorithm is Based on 6

Figure 1.2: Ant Will Always Choose the Shorter Path 6

Figure 1.3: Two Paths between the Ants' Nest and Food 7

Figure 1.4: Ants deposit Pheromones 7

Figure 2.1: A 532-city TSP Instance Created by Shen Lin of AT&T In 1987 11

Figure 2.2: A 2-opt Example 16

Figure 2.3: A 3-opt Example 16

Figure 2.4: A 2.5-opt Example 17

Figure 3.1: Program Flow of the Exhaustive Search 21

CHAPTER 1

INTRODUCTION

1.1 Heuristic Search

"Heuristic" is a word originally from the Greek word "eureka", which basically means "pertaining to finding". This word relates to many subjects of human interest. A simple example of heuristic is: if you can't find a solution guess one and explore its validity. ”Working backwards” in this way can often shed new light on a topic. This concept can be used in areas like computer science, psychology, philosophy, law, etc. The discussion here will deal only with its relation to computer science. In the field of computer science a heuristic is sometimes called an approximate algorithm. There are many different heuristic search algorithms such as hill climbing, best first search and A*.

A heuristic coding technique will usually generate good solutions, but doesn’t always find the best solution. It is a logic based algorithm which is designed to work quickly, but at the cost of providing the guaranteed best solution. In some cases it may fail altogether. A metaheuristic is a higher-level strategy that works by guiding multiple heuristics in a search for improved solutions. Ant System Metaheuristic is used to solve the Capacitated Vehicle Routing Problem (CVRP).

The underlying principle of a heuristic search is to try only the paths that seem to be getting us closer to our goal state, instead of trying all possible search paths. However at times it

can be difficult to detect which path is leading to the goal state, so some method of quantifying a given path's potential quickness is called for.

A heuristic search can estimate the next potential states' "closeness" to a goal state. The measure of closeness is numerical so they can be compared and the path with the best measure of closeness can be selected. This method is good for a very large search space where exhaustively searching the entire space would not be feasible. To measure closeness, an evaluation function is needed which can score a node in the search tree based on how close to the goal state it seems to be. At best this will always be just a guess, but a good guess.

For example, considering a minimum cost problem of driving from a starting city S to a goal city G (there are other cities between city S and city G), the straight line distance between a possible next city and the goal city could be used to evaluate the "closeness" of that candidate city. This method probably doesn't accurately reflect the actual distance when traveling because the roads are not always straight, and it doesn't take into account how this move will affect the travel time to other cities. Nonetheless it provides a quick way of guessing which helps guide the search.

For an NP-Hard problem heuristics are one kind of solution approach. There are two subcategories: constructive and local search methods. Constructive algorithms generate solutions from scratch by iteratively adding solution components to an initially empty solution until the solution is complete. A local search starts from some initial solution and repeatedly tries to improve the current solution by introducing local changes. The first step in a local search algorithm is to define a neighborhood structure over the set of candidate solutions. Then the *hill-climbing* or *iterative improvement* technique is used to find the local maxima as a current solution. For the *hill-climbing* technique the local maxima can be thought of as a hill peak in a

range of hills. The *hill-climbing* technique will find the tallest “hill peak” of the hills in a specified local search area, but that is not necessarily the highest peak in the entire hill range. The local maxima are not necessarily the true or global maxima. Note that any global maximum is also a local maximum; however a local maximum may or may not be the global maximum. The local search will repeatedly look for improved solutions and replace the current solution with improved solutions until no improved solutions can be found.

1.2 The Problem

To prevent the gypsy moth (Lymantria dispar L.) from devastating North America's forest, the US Forest Service has to spray pesticides by aircraft. Using programs to find the optimal spray planning will highly increase the spray productivity. At first, GypsES (Gypsy Moth Expert System) was developed as a computerized decision support system for assisting the pesticide management programs. Because GypsES used a geographical information system (GIS) framework for on-screen digitizing and aerial photographs, it was primarily used in the spray treatment planning by aircraft. However, GypsES could not determine the pesticide needed for an aerial spray treatment project. Determining pesticide needs for the treatment projects was mainly based on guesswork or historic information from other projects until CASPR (Computer Assisted Spray Productivity Routing) was developed in 1988. CASPR was used to implement the procedure for evaluating the efficiency of a single sprayed block. Later on, CASPER was developed as an improved version of CASPR. CASPER requires the scheduled route as an input and then reports statistics related to spray efficiency. The project in this paper emphasizes how to get a good scheduled route as a part of spray treatment planning (STP).

To apply the pesticide to the forest efficiently, planning the spray procedure to find the optimal route is critical. The spray planning problem was simplified as the CVRP, and a heuristic search was used to find the optimal spray route. Among several heuristic methods the ant metaheuristic, or Ant Colony Optimization (ACO), was implemented. The results from ACO were compared with the results from the exhaustive search and another heuristic method known as Genetic Algorithm (GA).

1.3 Ant Metaheuristic --- Ant Colony Optimization (ACO)

1.3.1 Metaheuristic

For a single-run with a heuristic algorithm constructive methods may get a limited number of different solutions, especially for greedy construction heuristics. A local search may stop at local optima which may be far from the solution we want. Metaheuristic methods are used to overcome these problems. The disadvantages of the heuristic methods can be counteracted by simply generating multiple heuristic runs. This is known as a metaheuristic approach. When an acceptable answer cannot be found with a heuristic approach, a metaheuristic method can be employed.

A metaheuristic is a set of algorithmic concepts that guides a series of heuristic methods. It could be considered to be a general-purpose heuristic method designed to guide a set of underlying problem-specific heuristics toward promising regions of the search space that contain high-quality solutions. This role makes it applicable to a wide set of different problems. It can be

thought of as a general algorithmic framework applied to different optimization problems with relatively few problem specific modifications. A local search or construction heuristic could be considered a metaheuristic method. Other examples of metaheuristics are tabu search, evolutionary computation, and ant colony optimization. Using metaheuristics can increase the probability of good solutions in a reasonable amount of time by making use of combinatorial optimization.

1.3.2 Ant Colony Optimization

Ant Colony Optimization (ACO) is a metaheuristic that uses "virtual" ants that cooperate to find good solutions to difficult discrete optimization problems. Its idea originated from biology research of social insect societies like ant colonies. In the real world ants coordinate their activities by stigmergy, which is a kind of indirect communication by leaving chemicals called pheromones. According to Deneubourg, Goss, and their colleagues' research (Deneubourg, Aron, Goss, & Pasteels, 1990; Goss et al., 1989) foraging ants can find the shortest path between their nest and a food source by marking the path they follow with pheromones.

The research on ants was conducted by carrying out a double bridge experiment. In this experiment there are two bridges between the ants' nest and the food source (see Figure 1.3). One bridge is longer than the other, and ants are free to move between the nest and the food. At first the path the ants choose seems random, but after a period of time the ants start to converge upon the shorter of the two bridges. The convergence of the ants upon the shorter path is an emergent behavior. While ants move they lay down pheromones in a quantity proportional to the quality of the food source discovered. Other ants observe the pheromone trail and are attracted to

follow it. Paths leading to rich nearby food sources will be more frequented and consequently the corresponding pheromone trails will grow faster.

The ACO model is based on this phenomenon. The basic ACO approach is to code "virtual" ants within the algorithm to cooperate and solve the routing problem by local sensing and depositing virtual pheromones along the paths, which are typically represented as arcs in connected graphs.

Figure 1.1: Ants are the Insects the ACO Algorithm is Based on

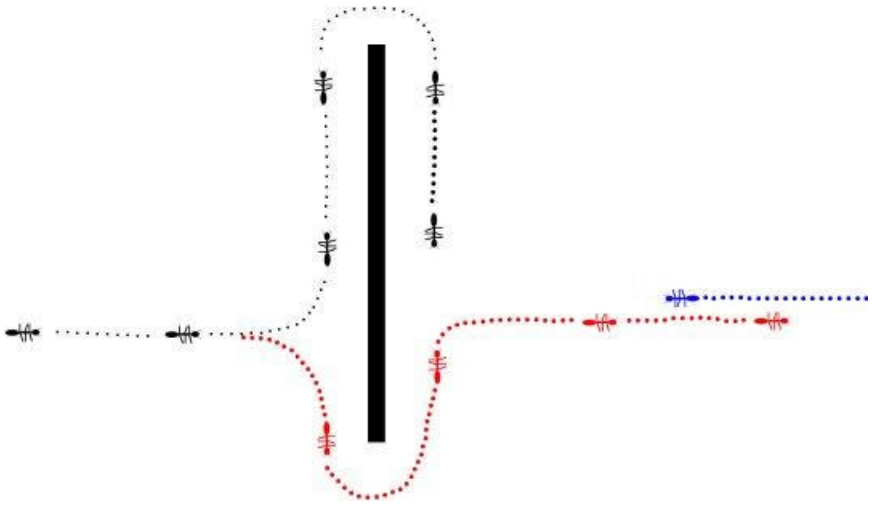

Figure 1.2: Ants Will Always Choose the Shorter Path

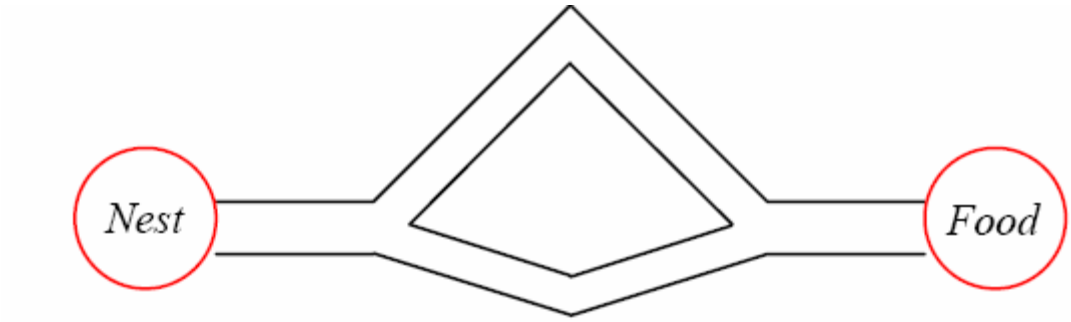

Figure 1.3: Two Paths between the Ants' Nest and Food

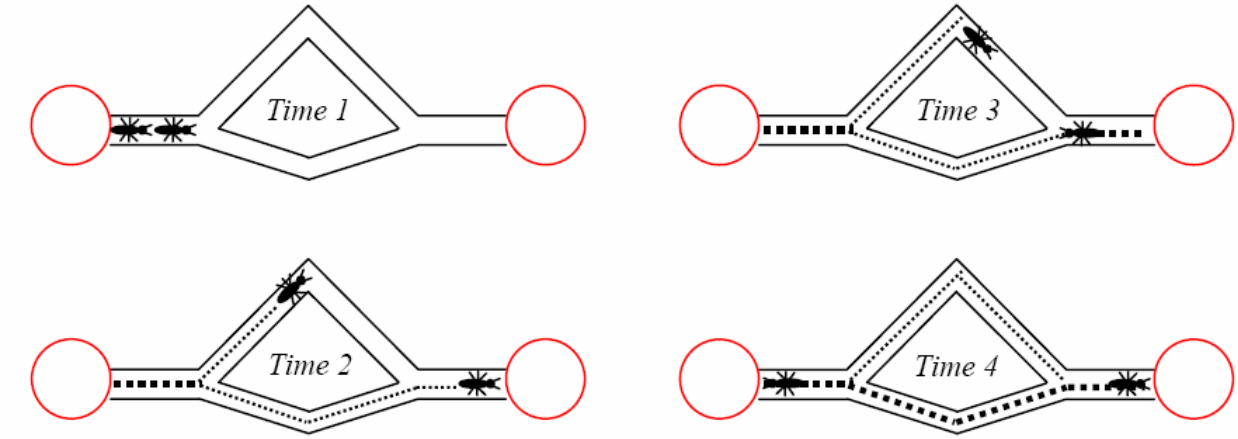

Figure 1.4: Ants deposit Pheromones

The first ant metaheuristic called Ant System (AS) was introduced by Marco Dorigo and colleagues (Dorigo M., 1992). Then Elitist Ant System (EAS) was introduced as the first improvement of the initial AS (M.Dorigo, V.Maniezzo, and A. Colorni., 1996). Bernd Bullnheimer, Richard F. Hartl, and Christine Strauss proposed another improvement for AS which is called Rank-Based Ant System (Bullnheimer B., R.F. Hartl and C. Strauss, 1999). Later on, Dorigo, Gambardella and Stützle worked out various extended versions of the AS paradigm. Dorigo and Gambardella have proposed Ant Colony System (ACS). Stützle and Hoos have proposed MAX-MIN Ant System (MMAS) (Stützle T. and H. Hoos., 1997). Dorigo,

Gambardella and Stützle have also proposed new hybrid versions of ant colony optimization which include local search (Dorigo M., Gambardella L.M., Middendorf M. and Stützle T., 2002).

Ant Colony Algorithms are typically used to solve minimum cost problems. The problem is usually represented as graphs of nodes and undirected arcs. The basic idea of the algorithm is that the virtual ants (referred to as ants hereafter) evaluate the cost of the arcs they have traversed by leaving pheromones on them, and eliminate any loops from their memorized arcs. In a given amount of time the shorter arcs will receive a greater deposit of pheromones. A pheromone evaporation rule will be included which will help remove some poor quality solutions. The ant's memory allows it to retrace the arc it has followed while searching for the destination node. At the beginning of the search process, a constant amount of pheromone is assigned to all arcs. When located at a node *i* an ant *k* uses the pheromone trail to compute the probability of choosing *j* as the next node. The ants will iteratively move from one node to next, causing increasing pheromone deposits while less frequented paths are subject to evaporation (this point will be explained in details in chapter 2). The more iterations, the better the solutions will be.

Below are the basic steps for solving a problem by ACO:

1) Represent the problem in the form of sets of components and transitions, or by a set of weighted graphs, on which "ants" can build solutions
2) Define the meaning of the pheromone trails
3) Define the heuristic preference for the ant while constructing a solution
4) If possible implement an efficient local search algorithm for the problem to be solved.
5) Choose a specific ACO algorithm and apply to the problem being solved
6) Tune the parameters of the ACO algorithm.

ACO can be used for both static and dynamic combinatorial optimization problems. It tends to perform better than other global optimization techniques such as simulated annealing for routing problems. Its theoretical analysis is difficult due to its experimental research nature, and its coding is somewhat complicated.

CHAPTER 2

BACKGROUND

2.1 Problem Domain

2.1.1 Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is stated as follows: given a number of cities and the costs of traveling from any city to any other city, find the cheapest round-trip route that visits each city exactly once and then returns to the starting city.

According to graph theory, the equivalent formulation is: given a complete weighted graph (where the vertices would represent the cities, the edges would represent the roads, and the weights would be the distances of the roads) find a Hamiltonian cycle with the least weight. A Hamiltonian cycle (or Hamiltonian circuit, vertex tour, graph cycle) is a cycle that visits each vertex exactly once, excluding the start/end vertex.

In the TSP problem it is required to return to the starting city. But even if we remove this requirement it does not change the computational complexity of the problem. It is said to be a NP-hard (Non-deterministic Polynomial-time hard) problem. To define the TSP problem as a NP-complete problem a threshold of cost needs to be determined. Any route that meets the stated requirements and has a cost below the threshold is a solution. An NP-hard problem is said to be at least as hard as any NP-complete problem. It is hypothesized that NP-complete problems could be solved in polynomial time on a deterministic Turing machine but this hypothesis has not been proven.

The problem is of considerable practical importance, and is common in transportation and logistics areas. A classic example is in printed circuit manufacturing: scheduling the route that the drill machine travels to drill holes in a printed circuit board (PCB). In robotic machining or drilling applications the "cities" are parts to machine or holes (of different sizes) to drill, and the "cost of travel" includes time for retooling the robot (single machine job sequencing problem). For the problem addressed in this paper the “cities” are blocks to be sprayed.

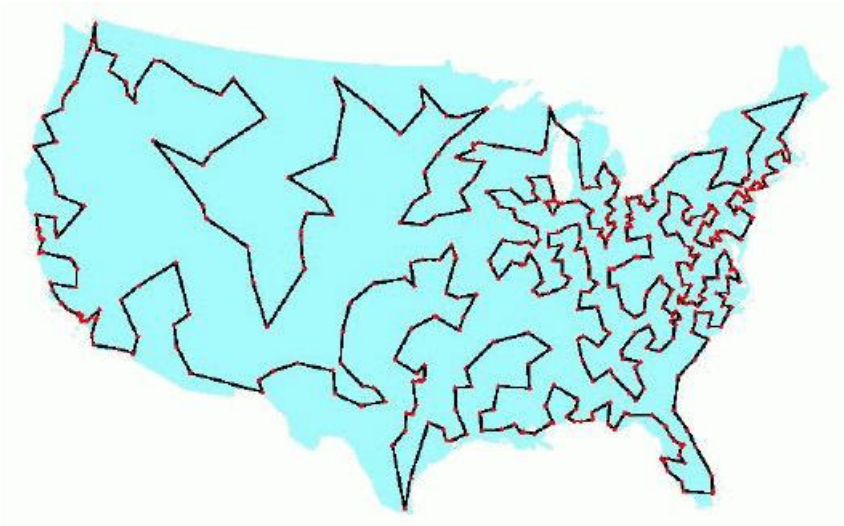

Figure 2.1: A 532-city TSP Instance Created by Shen Lin of AT&T In 1987

2.1.2 The Capacitated Vehicle Routing Problem

The Capacitated Vehicle Routing Problem (CVRP) is a subcategory of the Vehicle Routing Problem (VRP). The VRP combines an efficient set of multiple TSP routes to form an even more complex problem. In its simplest form the VRP involves some vehicles starting at a central depot, making some number of service stops, then returning to the central depot. The objective of the VRP is to minimize route length, service cost, travel time, or any combination of

these variables. The VRP has a few variants which can be divided into several subcategories including CVRP, VRP with Time Windows, Split Delivery VRP, and Multiple Depot VRP.

The CVRP is similar to the VRP but with the additional constraint that all vehicles within the fleet have a uniform carrying capacity of a single commodity. The commodity demand along any route assigned to a vehicle must not exceed the capacity of the vehicle assigned to that route. As an NP-complete problem, the CVRP is a common topic in operation research. It is also related to mathematics, graph theory and transportation science. Solutions to this problem can be found using heuristic methods such as Branch-and-Cut (or Branch-and-Bound) and genetic algorithms.

Of the many heuristic search methods used for this problem, a particularly well studied choice is the Branch-and-Cut method. This method is a decomposition-based separation methodology for capacity constraints that excels in solving small instances of the TSP efficiently. Specifically, when standard procedures fail to separate a candidate point it attempts to decompose it into a combination of TSP tours. Any successful tours are then examined to see if they violate capacity constraints. If there are not any successful tours the Farkas Theorem (T.K. Ralphs, L. Kopman, W.R. Pulleyblank & L.E. Trotter, 2003) uses a hyperplane to separate the point from the TSP polytope.

Genetic algorithm (GA) is also a powerful method, and an even faster hybrid GA is under development. For that developmental hybrid GA the Initialization Heuristics (IH) are used to generate an initial population, while the other two heuristics RemoveSharp and LocalOpt can be applied to the GA offspring obtained either by crossover or by shuffling (G. Andal Jayalakshmi, S. Sathiamoorthy, and R. Rajaram, 2001).

Metaheuristic methods are also good for both the TSP and the CVRP problems. The ant system metaheuristic combines an adaptive memory with a local heuristic function to repeatedly construct solutions to hard combinatorial optimization problems like the CVRP. This paper will deal with a metaheuristic method called Ant System Metaheuristic.

The problem this paper focuses on is a specific case of CVRP with a single aircraft. The specific case is stated as follows:

1. Let G=(V,E) be a connected graph, where V=$\{ V_1, V_2, V_3, \ldots, V_n \}$ is the block set and V_1 denotes the vertex at which the airport is located;
2. E=$\{(V_i, V_j), i \neq j \}$ is the sets of arcs between V_i and V_j. Each arc has an associated weight d_{ij}, which is the distance between V_i and V_j.
3. Let a non-negative value Q_i be the load associated with V_i. The load is the amount of pesticide that the given block needs.

It was assumed that there is a fleet of homogeneous aircraft with pesticide tank capacity Q based at the airport and available. A trip is everything that occurs between the time the aircraft takes off from the airport and the time it returns to airport. A trip of spraying blocks means that an aircraft travels among several blocks and sprays them without going back to the airport to refill pesticide. The CVRP in our project consists of determining aircraft spray routes starting and ending at the airport such that:

- The load associated with any given block is sprayed by exactly one aircraft.
- The sum of all loads in one trip of spraying blocks does not exceed Q.

The linear combination of the number of aircraft and the total distance traveled by those aircraft was minimized.

2.2 Detailed Ant System Metaheuristic

The basic steps of Ant System are initialization, generate new solution, local search, and update pheromones. During initialization the distance matrix, pheromone matrix, a matrix with nearest neighbor lists of depth *nn*, a choice information matrix with combined pheromone and heuristic information, and the ant structure are created. The distance matrix contains the distances between each block. For example, distance_matrix[i][j] is the distance between block i and block j. Each block has its own nearest neighbor list which holds the *nn* blocks near it. Typically, depth *nn* is a small number ranging between 15 and 40 if the number of total blocks is bigger than 40. If the total blocks are less than 15 then depth *nn* is less than or equal to the number of total blocks. To get the nearest-neighbor list for a block i the first step is to sort the distance list d_i in order of increasing distance to get $d_i^{'}$. When sorting d_i the order of blocks at equal distance to block i does not matter and may be chosen randomly. The next step is to put the index of the r-th nearest block to block i in a matrix nn_list[i][r]. If the r-th nearest block to block i is block j, nn_list[i][r]=j.

After initializing a pheromone matrix, a choice information matrix, and an ant structure, ant tours can be constructed. The first step is to mark all blocks as unvisited. The next step is to assign each ant an initial block randomly, then assign them a complete tour using the ant system action choice rule. After that the ants are moved back to the initial block and each ant's tour length is computed. Below is the equation for the pertinent rule. p_{ij} is the probability of selecting block V_j while the current ant is at block V_i. If $j \notin N_i^k$, the value of p_{ij} is set to be 0. τ_{ij} indicates how many local pheromones are currently deposited along arc (V_i, V_j). η_{ij} is defined

as the reciprocal of the cost along arc (V_i, V_j). Thus $\eta_{ij} = 1/d_{ij}$, where d_{ij} is the distance between block V_i and V_j. α and β are biases to be adjusted as needed. They are parameters which determine the relative influence of pheromone trails and η_{ij} respectively.

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{ij}]^\alpha [\eta_{ij}]^\beta}, \text{ if } \quad j \in N_i^k ,$$

The tour construction steps are repeated until a tour has been completed by all ants. After the solutions are constructed, a local search is used to improve the solutions. The local search is a general approach for hard combinatorial optimization problems. A local search tries to improve the current solution by local changes that iteratively explore neighborhoods of solutions. The performance of a local search algorithm is affected primarily by the choice of an appropriate neighborhood structure. A neighborhood structure is represented by the function $N : S \mapsto 2^s$ which assigns a set of neighbors $N(s) \subseteq S$ to every $s \in S$. $N(s)$ is also called the neighborhood of s.

Explicitly going through the set of all possible neighbors is difficult. A simple way of defining neighborhood structure is to implicitly define possible local changes that may be applied to arrive at a solution. That solution is a locally optimal but not guaranteed to be globally optimal. The rules that govern the choice of acceptable neighborhood solutions are the best-improvement rule and the first-improvement rule. The best-improvement rule means to choose the neighborhood solution giving the best improvement of the objective function. The first-improvement rule chooses the first improved solution. There are three popular types of local search: 2-opt, 2.5-opt and 3-opt.

2-opt can also be called 2-exchange. In the TSP case with a known candidate solution, s, the 2-exchange neighborhood of a candidate solution s consists of the set of all the candidate

solutions s' that can be obtained from s by exchanging two pairs of arcs in all possible ways. A specific 2-opt example is shown in Figure 2.2 in which a set of arcs (b,c) and (a,f) are replaced by a different set of arcs (a,c) and (b,f). The 3-opt neighborhood consists of those tours that can be obtained from a tour s by replacing at most three of its arcs. Removing three arcs will generate three partial tours. Recombining three partial tours will produce a full tour in eight different ways. The 2.5-opt local search is a 2-opt local search under a restricted version of a 3-opt move. While checking whether the 2-opt move will result in a better tour, it is also checking for an improved move by inserting the city between a block, i, and its successor. Specific examples of these local search algorithms are shown in the figures below.

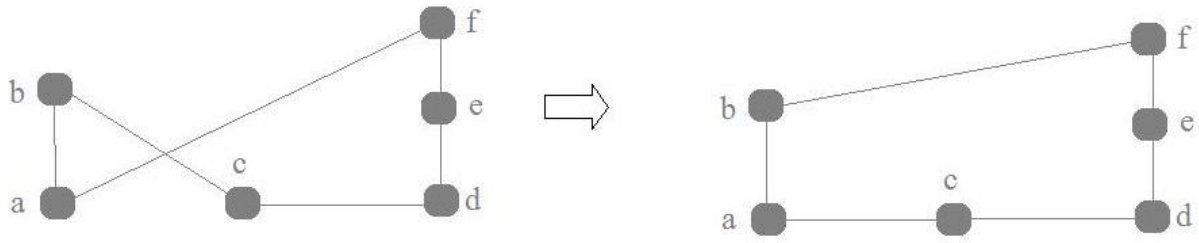

Figure 2.2: A 2-opt Example

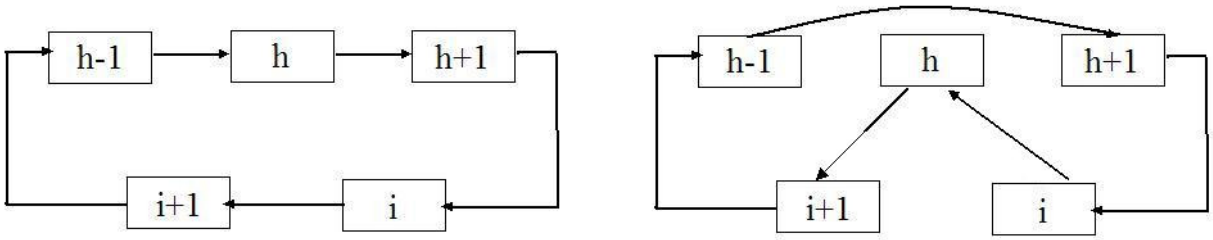

Figure 2.4: A 2.5-opt Example

After the local search step is complete, the next and usually last step of the Ant System Algorithm is to update pheromones. There are two sub-procedures in the updating pheromone step: pheromone evaporation and pheromone deposit. Pheromone evaporation decreases the value of the pheromone trail on all the arcs (i,j) by a constant factor ρ. Then pheromone is deposited, or added, to the arcs belonging to the tours constructed by the ants. The global updating rule in for the Ant System Algorithm can be found below. Ψ is the global best routing solution found so far. ρ is within the range of (0,1] and indicates the persistence of the pheromone while (1- ρ) indicates the evaporation of the pheromone. C_{gb} is the cumulative cost of all routes in Ψ.

$$\tau_{ij}^{new} = (1-\rho)\tau_{ij}^{old} + \rho\Delta\tau_{ij}$$

$$\Delta\tau_{ij} = (C_{gb})^{-1}, \text{ if } (V_i, V_j) \in \Psi$$

2.2.1 Rank-Based Ant System

The difference between the original Ant System and Rank -Based Ant System is the method of updating pheromone trails. In this algorithm ants are ranked based on the quality of their solution, and pheromone deposit corresponds to the rank of the ant. Table 2.1 shows the difference between the two algorithms' parameters. α, β and ρ are discussed earlier in this

chapter. τ_0 is the initial value for all τ_{ij}. m is the number of ants and C^{nn} is the length of the tour generated by the nearest-neighbor heuristic. n is the number of blocks and r is the ranking index.

ACO algorithm	α	β	ρ	m	τ_0
AS	1	2 to 5	0.5	n	m/C^{nn}
AS-rank	1	2 to 5	0.1	n	$0.5r(r-1)/\rho\ C^{nn}$

Table 2.1: Parameter Setting for ACO Algorithms without Local Search

The ants should be sorted by increasing tour length and the quantity of pheromone. An ant's deposits are weighted according to the rank of the ant. These procedures should be done before updating the pheromone trails. If there is a tie between two, randomly order them. Only the w-1 best-ranked ants and the ant that produced the best-so-far tour can deposit pheromone. The ant which generated the best-so-far tour does not have to be among the set of ants of the current algorithm iteration. The weight of the best-so-far tour is w, which is typically 6. Thus the best-so-far tour's contribution $1/C^{bs}$ is multiplied by w. The r-th ranked best ant of the current iteration deposit with a value $1/C^{r}$ multiplied by a weight given by the biggest number among {0, w-r}. Below is the formula for the Rank-Based Ant System pheromone update rule. C^{bs} is the length of the best-so-far tour. The best-so far tour is the best tour found since the start of the algorithm.

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{r=1}^{w-1}(w-r)\Delta\tau_{ij}^{r} + w\Delta\tau_{ij}^{bs},$$

Where:

$$\Delta\tau_{ij}^{r} = 1/C^{r}$$

$$\Delta\tau_{ij}^{bs} = 1/C^{bs}$$

CHAPTER 3

METHODOLOGY

3.1 Approach

This project utilizes two phases with each phase using different methods. The emphasis is on phase 2 while phase 1 is just used to get the real optimal route for comparison. Phase 1 uses an exhaustive search, i.e. greedy search, to find the minimum total ferry distance. Because the total spray time is supposed to be non-variant regardless of how the route changes, the total flight time will be determined by the total ferry distance. Thus the total ferry distance is the criteria to choose the best route in the exhaustive search results. After reading the data files and performing initialization, the program will generate a permutation list of all the blocks. In the next step, assuming fuel is sufficient to fly to all the blocks, the exhaustive search tries every possible route and reports the one with minimum total ferry distance as the solution. Then the total fight time of the optimal route is calculated and the program will output the route with the time and distance statistics.

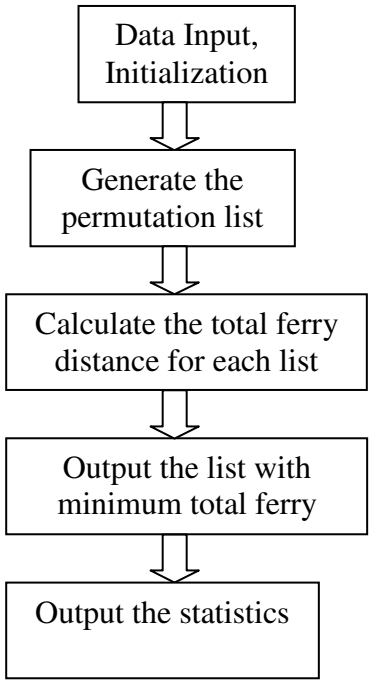

Figure 3.1: Program Flow of the Exhaustive Search

Both the exhaustive search method and the ACO method use the same way to calculate the total flight time. The total flight time consists of two parts: the total spray time and the total ferry flight time. The total spray time is calculated by the formula:

$$TotalSprayTime = \sum_{i=1}^{N} (s-1) \times TurningTime + s \times length_i \div SpraySpeed$$

Where N is the number of blocks, $length_i$ is the length of the long side of the i-th block (every block is assumed rectangular), and $s = \frac{Width_i}{SwathWidth}$. $Width_i$ is the short side rectangular width of the i-th block

The total ferry time is determined by the total ferry distance since the ferry speed is a constant. The total ferry distance is accumulated in every single trip. Every single trip refers to a trip between two blocks or a trip between a block and the airport. If the current pesticide carried

by the aircraft is more than the pesticide needed by current block, only the distance between the last block and the current one will be accumulated. Otherwise, the trips back to the airport need to be included. Originally the needed pesticide was calculated by the block area, taking into account excluded areas such as lakes:

$$\text{Pesticide Needed(gal)} = \text{pesticide application rate(gal/ac)} \times \text{the block area accounting for exclusions(ac)}$$

The result from the formula above is listed in Table 3.1. As the program evolved a new equation for pesticide needed was used that ignored areas that could be excluded from spraying such as lakes:

$$\text{Pesticide Needed(gal)} = \text{pesticide application rate(gal/ac)} \times \text{the block area ignoring exclusions(ac)}$$

The results from this formula can be found in Table 3.2. Comparing the current result from Table 3.2 with the previous one in Table 3.1 shows that block 7 needs less pesticide when the excluded areas were not counted in the spray area. The difference will result in different total spray times. The method shown in Table 3.2 is more practical and accurate. Therefore it is chosen to calculate the needed pesticide for every block.

Block ID No.	Pesticide Needed(gallon)
1	316.47
2	201.96
3	415.47
4	409.86
5	489.06
6	1021.02
7	143.55
8	65.34
9	48.18
10	68.97
Total Pesticide Needed	3179.88

Table 3.1 Needed Pesticide calculated by the area accouting for exclusions

Block ID No.	Pesticide Needed(gallon)
1	316.47
2	201.96
3	415.47
4	409.86
5	489.06
6	1021.02
7	136.29
8	65.34
9	48.18
10	68.97
Total Pesticide Needed	3172.62

Table 3.2 Needed Pesticide calculated by the area ignoring exclusions

The total flight time is calculated by the following formula:

$$\text{Total Time} = \frac{\text{Total Ferry Distance}}{\text{Ferry Speed}} + \text{Total Spray Time}$$

Phase 2 uses the ant metaheuristic to find the route with a nearly minimum total ferry distance. A rank-based ant system algorithm with static input data array is implemented in the program. Its model is a connected graph with blocks as nodes and routes between each node as arcs. After an iteration of ant tour constructing is finished results from a good tour will be recorded and compared with the results from last iteration. The tour with shorter total ferry distance will be claimed as the best-so-far tour. The program will continue to run iterations repeatedly and output the last best-so-far tour as the nearly optimal solution. The parameter values are decided by previous experiments by others researchers (recall from Chapter 2.2.1). Below is the ACO program module for phase 2:

```
/*
Construction graph:
It comprises one component for each of the spray blocks and arcs weighted by distance.
Constraints:
Each block can be visited once and the vehicle capacities cannot be exceeded.
Pheromone trails:
Pheromone trails πij are associated only with connections.  The
pheromone trail refers to the desirability of visiting block j directly
after i.
```

```
*/
/*main*/
Procedure ACOforSTP
InitializeData
    While (termination condition not met) do
/*The termination condition can be that the program has found a solution within a predefined
distance from a lower bound on the optimal solution quality, a maximum number of tour
constructions has been reached or a maximum number of algorithm iterations has been reached*/
        ConstuctAntSolutions
        ApplyLocalSearch
        UpdateStatistics
        UpdatePheromones
    End
EndProcedure

/*sub-procedure*/
Procedure InitializeData
    ReadInstance  /*read data for blocks and the aircraft */
    ComputeDistances  /* create the distance matrix*/
    ComputeNearestNeighborLists /* create a matrix with nearest neighbor lists of depth nn */
    ComputeChoiceInformation /* create pheromone matrix, create choice information matrix
with combined pheromone and heuristic information */
    InitializeAnts/* create ant structure, every ant has a memory storing tours, last tour length
and visited blocks*/
```

```
Procedure UpdatePheromones /*update pheromones by the depositing pheromone formula and
evaporation rules*/
    Evaporate
        For i=1 to n do
            For j=i to n do
                Pheromone[i][j]←(1-p)· Pheromone[i][j]
                Pheromone[j][i] ← Pheromone[i][j]
                endFor
        endFor
    endProcedure
for k=1 to m do
DepositPheromone(k)
endFor
ComputeChoiceInformation
endProcedure
```

3.2 Experiment Setup

The phase 1 program was done in Java. The exhaustive search features a very long runtime. For the block file used in the program, there are ten blocks. Thus the search will go through 3,628,800 (10!= 3628800) possible tours (the permutation of the ten blocks) to find the optimal solution. An ordinary computer with one 1.4-GHz CPU may run it for an extended length of time. In this case the program was run by a Multi-User Computing Server called Darwin in the Artificial Intelligence Center at UGA. Darwin has two Intel Xeon 3-GHz CPUs with 2 instruction pipelines. The Windows XP Operation System sees it as a 4-CPU machine and will automatically distribute the work among the CPUs. The runtime of the entire program was several minutes when run by Darwin. The results will be stated in next chapter.

Phase 2 was done in Visual Basic 6. This program runs much faster than the exhaustive search. An ordinary computer with one 1.4-GHz CPU can run it in seconds. There were a few assumptions made to make the problem easier to program. First of all the assumption was made that there is only one aircraft and one airport. Second, the aircraft will always spray an entire

block before moving on to the next block. So it is impossible to spray one block then stop when not finishing the whole block and spray another one. Third, the shape of every block is similar to a rectangle whose length is the block's minimum bounding rectangle's length (the longest side in any direction) and its width is the block's minimum bounding rectangle's width (shortest side in any direction). Length and width are measured in units of feet. The length and width data in the program is not the same "length" and "width" value in the original block data file. For every block, the "length" and "width" value are compared and the longer side is set to be the "length" in the program. For most of blocks in the original block data file, the length is longer than the width. But there is a block with its width longer than its length. Fourth, the aircraft will always spray along the long side. Fifth, the aircraft will always refuel completely and refill pesticide when returning to the airport.

The parameter setup is stated as following. α is 1, β is 2 and ρ is 0.1. w, which is the weight of the best-so-far tour, is 6 (recall from chapter 2). The number of ants is the same as the number of blocks since there are relatively few blocks. The number of nearest neighbors is set to be the same as the number of blocks as well. The algorithm can run for several iterations to return a good solution, but the maximum iteration number is set to be 10. Among the three kinds of local search, 2-opt is chosen since there are only a few blocks. All the statistics such as the total flight time are calculated the same way they were for the exhaustive search method.

CHAPTER 4

RESULTS

The exhaustive search and the ACO produce different route planning. Both are different from the results of a GA program written by another colleague. At first comparing different spraying plans and finding the optimal is difficult because statistics such as the total flight time are calculated in different ways in different programs. As a result comparisons between different route plans are difficult. To achieve the same assumptions and method to calculate the statistics, a program in Java was written. When inputting the spray plan of blocks excluding airport visits, the program will output the plan including airport visits and it gives the total fight time as well. It reported the same route planning as GA's result but a different total fight time. Detailed results are listed in the appendix. Some statistics comparisons are listed in Table 4.1.

In fact, CASPER can compute the statistics when inputting the route excluding airport visits. Table 4.2 shows the statistics for each method generated from CASPER. The total product (the total needed pesticide) is 3173 gallons. It is rounded from the result of 3172.62 gallons (the result from Chapter 3, Table 3.2). Comparing the three cases between Table 4.1 and Table 4.2, the total ferry time is exactly 3 hours different in each case. This difference is likely because CASPER is a higher fidelity program and makes different assumptions. However, that difference is not important because both tables show the same trend across the three methods. For both CASPER and the Java program the total flight time of GA is greater than that of ACO, which is greater than that of the exhaustive search (the method of computing the total flight time

used in ACO and the exhaustive search is shown the appendices). CASPER has a very advanced GUI compared to the Java program, which made it a major tool for computing the route statistics.

	GA	ACO	Exhaustive Search
Route (block only)	{6,3,4,5,7,8,10,9,2,1 }	{10,9,8,7,5,4,1,2,3,6}	{1,2,4,5,7,8,9,10,6,3}
The Total Ferry Distance (mile)	496.0	460.6	411.5
The Total Flight Time (hour)	12.6	12.3	11.9

Table 4.1 Statistics Comparisons by Java Program

	GA	ACO	Exhaustive Search
Route (block only)	{6,3,4,5,7,8,10,9,2,1 }	{10,9,8,7,5,4,1,2,3,6}	{1,2,4,5,7,8,9,10,6,3}
The Total Flight Time (hour)	15.6	15.3	14.9
Total Product (gal)	3173	3173	3173

Table 4.2 Statistics Comparisons by CASPER

Comparing those results, the exhaustive search gives the optimal route with the minimum total flight time 11.9 (Java program)/14.9 (CASPER) hours. ACO and GA give nearly optimal solutions but not the optimal route. Another point to keep in mind is, GA and ACO may use different criteria for searching their solutions. It indicates they may search for different search space. ACO searches for the minimum total ferry time without visiting the airport and then computes the reported efficient route adding necessary visits to the airport. In the future, ACO could be improved to search for the minimum total ferry time with visiting the airport. GA can search with fuel constraint or without it. CASPER does not search for the optimal route. It only calculates the statistics after inputting the routes manually.

CHAPTER 5

SUMMARY

5.1 Conclusions

This project has been presented as an application of the ant system metaheuristic to solve the route planning part of the Spray Treatment Planning (STP) problem. The STP problem is to plan the most efficient way to spray the forest consisting of several blocks of trees so that the flight time is minimized and to assist managers in evaluating the productivity and efficiency of different spray treatment projects. Because of the limitation of aircraft pesticide tank capacity, an aircraft may require multiple trips back to the airport to spray a treatment area (a block). Different spraying routes may result in different total flight times. The goal is to find spraying routes that improve the efficiency of the project.

A simplified model of the problem in the project is the CVRP. A typical CVPR may have multiple vehicles/aircrafts to serve. In our experiment, we assume there is only one aircraft to spray all blocks and there is only one airport. The straightforward way to solve the problem is to test every possible route and choose the one with minimum time. This is called the greedy approach, i.e. exhaustive search. When the number of blocks is large enough, this approach takes a very long time to convergence. When dealing with a large number of blocks it is impractical to compute every possible route, so a heuristic search is needed. Among a lot of heuristic and/or metaheuristic methods, the ant system metaheuristic is an efficient algorithm for minimum cost problems.

The work done for this project was primarily coding spray scheduling by ACO and an exhaustive search. The ACO program was written in Visual Basic 6 and tested with stable results. The exhaustive search program was written in Java and tested successfully. A small Java program was also written to calculate the statistics for all methods. Using that program the spray efficiency statistics for the whole set of blocks was obtained for comparison. The aim of the work done here was to develop a decision making program to aid CASPER by adding a scheduling model. The objective of the scheduling model is to get a good spray route so that the aircraft can accomplish its goal in the least amount of time. The shape of the block can influence the spray efficiency, and that effect has not yet been taken into account. Getting the minimum total ferry distance was the main goal of the work done in this paper, with the assumption that every block was rectangular.

According to the experiment, the Rank-Based Ant System based on ACO gives a good solution for ten blocks and one airport with less than a minute of runtime. The result was different from the result of GA, but the difference does not mean something is wrong with either method because of the nature of a heuristic search. A Heuristic search does not guarantee a true optimal solution.

GA and ACO reported different routes. ACO and the exhaustive search used the same method for computing total flight time, and were therefore easier to compare. Using the same program to compute the statistics for different routes would make comparison between GA and ACO easier. This was done by inputting the routes reported by GA, ACO and the exhaustive search, into a separate program coded in Java written to report statistics. Afterward CASPER was used to the same end. Using CASPER as a common ground made comparison much easier,

even more so because CASPER provides a simple GUI. The comparison of statistics shows that the total flight times for ACO and GA are similar and they both can be considered good solutions. However neither gave the best solution, which was the solution given by the exhaustive search.

5.2 Future Directions

This experiment is primarily based on minimum total ferry distance without visiting the airport. Adding the back trip to the airport cannot be implemented with a static data input. The result could be improved if a dynamic data input could be achieved. Then the airport could be added as a node in the CVRP model, and when the current pesticide carried by the aircraft is less than the pesticide needed by the next block the algorithm could direct the plane to the airport node. In the future, more constraints can be added and the distance array for ACO can be made dynamic so that the airport is included in the array when the aircraft has to refill spray pesticide. There are many other ACO algorithms to choose from and this experiment concentrated only on the Rank-Based Ant System. Other ACO algorithms could be run and benchmarked to get better solutions. Even beyond ACO other heuristic searches such as the tabu search could be attempted for comparison.

REFERENCES

Detailed Explanation of Traveling Salesman Problem: http://www.tsp.gatech.edu/index.html

http://en.wikipedia.org

Dorigo, M. (1992). *Optimization, Learning and Natural Algorithms* [in Italian]. PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Milan

Dorigo, M., Maniezzo, V., & Colorni, A. (1991). *Positive feedback as a search strategy. Technical report 91-016*, Dipartimento di Elettronica, Politecnico di Milano, Milan

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). *Ant System: Optimization by a colony of cooperating agents*. IEEE Transactions on Systems, Man, and Cybernetics, Part B, 26(1), 29-41

Dorigo M. & L.M. Gambardella. (1997). *Ant Colonies for the Traveling Salesman Problem.* BioSystems, 43:73-81.

Dorigo M., G. Di Caro & L. M. Gambardella. (1999). *Ant Algorithms for Discrete Optimization*. Artificial Life, 5(2):137-172.

Stützle T. and H. Hoos. (1997). *The MAX-MIN Ant System and Local Search for the Traveling Salesman Problem*. Proceedings of ICEC'97 - 1997 IEEE 4th International Conference on Evolutionary Computation, IEEE Press, 308-313.

Stützle T. and H. Hoos. (1997). *Improvements on the Ant System: Introducing the MAX-MIN Ant System*. ICANNGA97 - Third International Conference on Artificial Neural Networks and Genetic Algorithms, University of East Anglia, Norwich, UK, Wien: Springer Verlag.

Bullnheimer B., R.F. Hartl and C. Strauss. (1999). *An Improved Ant system Algorithm for the Vehicle Routing Problem*. Annals of Operations Research, Vol89, Number 0, 319--328

Dorigo M., Gambardella L.M., Middendorf M. and Stützle T. (2002). *Guest editorial: special section on ant colony optimization*. Evolutionary Computation, IEEE Transactions, Vol6, Issue 4, 317- 319

Karl F. Doerner, Richard F. Hartl, Guenter Kiechle, Maria Lucka, and Marc Reimann. (2004). *Parallel Ant Systems for the Capacitated Vehicle Routing Problem*. EvoCOP 2004, LNCS 3004, 72--83

G. Andal Jayalakshmi, S. Sathiamoorthy, and R. Rajaram. (2001). *A Hybrid Genetic Algorithm-A New Approach to solve Traveling Salesman Problem*. International Journal of Computational Engineering Science, Vol.2 No.2, 339--355

T.K. Ralphs, L. Kopman, W.R. Pulleyblank & L.E. Trotter. (2003). *On the capacitated vehicle routing problem. Mathematical Program.*, vol. 94, 343-- 359

W.D.Potter, etc. (2004) STP*: An Aerial Spray Treatment Planning System*, Technical report, Artificial Intelligence Center, University of Georgia, Athens, GA

Deneubourg, J.L., Aron, S., Goss, S., & Pasteels, J.M. (1990). *The Self-organizing Exploratory Pattern of The Argentine Ant*. Journal of Insect Behavior*,* 3, 159-168

Goss, S., Aron, S., Deneubourg, J.L., & Pasteels, J.M. (1989). *Self-organized shortcuts in the Argentine Ant*. Naturwissenschaften*,* 76, 579-581

Marco Dorigh, Thomas Stutzle. (2004) *Ant Colony Optimization*. The MIT Press, 2004

Garey, Michael R., Johnson, David S. (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman New York, 1979

Michael Rice. (2005) *A New Hybrid Computational Intelligence Algorithm for Optimized Vehicle Routing Application in Geographic Information Systems*, PhD thesis, Department of Geography, University of Georgia, Athens, GA

Potter, W.D., Deng, X., Li, J. (2000) *A web-based expert system for gypsy moth risk assessment,* Computer and Electronics in Agriculture, 27(2000): 95-105.

Shoshana, A., Julien, B. (1999). *Approximation algorithm for the capacitated traveling salesman Problem with pickups and deliveries,* Naval Research Logistics, 46: 655-670.

Potter, W.D., Ramyaa, Li, J., Ghent, J., Twardus, D., Thistle, H. (2002). STP: an aerial spray treatment planning system, SoutheastCon, 2002. Proceedings IEEE, 300-305.

Formal definition of NP-completeness: http://en.wikipedia.org/wiki/NP-complete

APPENDICES

A Flight Plan Result from ACO code

"Air Tractor 400(App. Rate=.33(gal/ac),Avg. Load=350 (gallons))"

"Block data source: sample_blockdata.txt"

start from the airport

go to block 10.0

go to block 9.0

go to block 8.0

go to block 7.0

go to block 5.0

go to airport

go to block 5.0

go to airport

go to block 5.0

go to block 4.0

go to airport

go to block 4.0

go to block 1.0

go to airport

go to block 1.0

go to block 2.0

go to block 3.0

go to airport

go to block 3.0

go to airport

go to block 3.0

go to block 6.0

go to airport

go to block 6.0

go to airport

go to block 6.0

go to airport

go to block 6.0

go to airport

the total ferry distance is 460.5900000000001

the total time is 12.28825 hours

B Date File Used

Block data source

1,483003.2293,4116981.4864,959,0,8024,5524,28.63,0.00,12.64,24.98,36.00,45.27,44.87,56.08,
57.32,58.29,59.12

2,502251.0036,4110413.8127,612,0,5301,5212,16.35,12.64,0.00,13.00,25.29,32.80,32.29,43.46,
44.70,45.66,46.49

3,523008.3857,4113061.3787,1259,0,9466,5911,3.90,24.98,13.00,0.00,23.05,23.76,20.56,32.83,
34.01,34.25,34.93

4,523943.3668,4075980.7562,1242,0,11005,5806,21.43,36.00,25.29,23.05,0.00,15.81,22.96,27.0
4,28.22,31.16,32.38

5,548039.4700,4084152.4641,1482,0,11889,5556,20.19,45.27,32.80,23.76,15.81,0.00,9.66,11.67
,12.91,15.43,16.62

6,552922.0797,4098916.0143,3094,0,13540,11209,16.67,44.87,32.29,20.56,22.96,9.66,0.00,12.8
9,13.93,13.69,14.37

7,566801.9919,4083496.0452,435,22,5360,4735,28.97,56.08,43.46,32.83,27.04,11.67,12.89,0.00
,1.26,4.77,6.04

8,568794.0784,4083088.2945,198,0,3503,2984,30.13,57.32,44.70,34.01,28.22,12.91,13.93,1.26,
0.00,4.16,5.35

9,572443.1959,4088696.0940,146,0,3705,2536,30.35,58.29,45.66,34.25,31.16,15.43,13.69,4.77,
4.16,0.00,1.28

10,574180.4606,4089813.1696,209,0,3802,2730,31.04,59.12,46.49,34.93,32.38,16.62,14.37,6.04
,5.35,1.28,0.00

==========

Aircraft details:

AirTractor AT400

.33,"App. Rate (gal/ac)"

125,"Swath Width (ft)"

140,"Application Speed (mph)"

120,"Ferry Speed (mph)"

30,"Avg. Turning Time (seconds)"

350,"Avg. Load (gallons)" - pesticide capacity

C Part of program in Phase 2

```
'The method of computing the total flight time used in ACO.
Dim i, acoOutput As Long
Dim totalSprayConsumption, curpest, remainedPest As Double
Dim ToAirportNo, AirportLabel, airport_visits As Integer
Dim t1, t2 As Double
Dim TotalFerryDistance As Double
Dim totalSpraytime As Double
Dim currentdir As String
TotalFerryDistance = 0
totalSpraytime = 0
totalSprayConsumption = 0
airport_visits = 0
```

```
acoOutput = FreeFile
currentdir = CurDir$ + "\AcoFileOutput.txt"
Open currentdir For Append As #acoOutput
Write #acoOutput, "Air Tractor 400(App. Rate=.33(gal/ac),Avg. Load=350 (gallons))"
Write #acoOutput, "Block data source: sample_blockdata.txt"
Write #acoOutput, " "
Write #acoOutput, "Start from airport"

AirportLabel = 1
curpest = aircraft.AverageLoad    'curpest is the current spray in the aircraft tank

For i = 0 To blockno_n - 1
t1 = instance.Ewidth(i) / aircraft.SwathWidth 't1 is the number of swath. unit: ft
totalSpraytime = totalSpraytime + (t1 - 1) * aircraft.TurningTime / 3600 + t1 *
instance.Elength(i) * 0.00019 / aircraft.SpraySpeed
'TurningTime unit :sec,Elength unit: ft,SpraySpeed:mph
'1sec=1/3600 hour,1 foot = 0.000 189 394 mile

If i = 0 Then
TotalFerryDistance = instance.ToAirportDistanc(best_so_far_ant.tour(i))
End If

'If the current spray is enough for the current block, the aircraft don't fly to the airport.
```

```
 If (curpest > instance.SprayConsumption(best_so_far_ant.tour(i))) Then
 Write #acoOutput, "Go to Block ID No." & (best_so_far_ant.tour(i) + 1)
Write #acoOutput, " "
        curpest = curpest - instance.SprayConsumption(best_so_far_ant.tour(i))


            If i = 0 Then
TotalFerryDistance = instance.ToAirportDistanc(best_so_far_ant.tour(i))
            Else
'instance.distance(k) is the distance between last block and the current block
TotalFerryDistance = TotalFerryDistance + instance.distance(best_so_far_ant.tour(i - 1),
best_so_far_ant.tour(i))
            End If
    Else
Write #acoOutput, "Go to Block ID No." & (best_so_far_ant.tour(i) + 1)
'If the current spray is less than the block need, calculate the trips to the airport as airport_visits
        airport_visits = Int((instance.SprayConsumption(best_so_far_ant.tour(i)) - curpest) /
aircraft.AverageLoad) + 1
             For ToAirportNo = 1 To airport_visits
            Write #acoOutput, "Go to airport"
            Write #acoOutput, "Go to Block ID No." & (best_so_far_ant.tour(i) + 1)
            Next
        'After refilling spray then applying to the current airport, calculate the spray left-over
```

```
remainedPest = (instance.SprayConsumption(best_so_far_ant.tour(i)) - curpest) Mod
aircraft.AverageLoad
        curpest = aircraft.AverageLoad - remainedPest
            If i = 0 Then
TotalFerryDistance = instance.ToAirportDistanc(best_so_far_ant.tour(i)) * (2 * airport_visits + 1)
            Else
TotalFerryDistance = TotalFerryDistance + instance.distance(best_so_far_ant.tour(i - 1),
best_so_far_ant.tour(i)) + instance.ToAirportDistanc(best_so_far_ant.tour(i)) * 2 * airport_visits
            End If
    End If
Next
TotalFerryDistance = TotalFerryDistance + instance.ToAirportDistanc(best_so_far_ant.tour(i))
t2 = totalSpraytime + TotalFerryDistance / aircraft.FerrySpeed 't2 unit :hour
Write #acoOutput, "go to airport "
Write #acoOutput, " "
Write #acoOutput, "The Total Ferry Distance is (unit:mile) " & TotalFerryDistance
  Write #acoOutput, " "
  Write #acoOutput, "The Total Spray time is (unit:hour) " & totalSpraytime
  Write #acoOutput, " "
 Write #acoOutput, "The total time is (unit:hour) " & t2
  Write #acoOutput, " "
Close #acoOutput
MsgBox "the program is finished", vbOKOnly
```