3-2002

A Multiple Ant Colony Metaheuristic for the Air Refueling Tanker Assignment Problem

RonJon Annaballi

A MULTIPLE ANT COLONY METAHEURISTIC FOR THE AIR REFUELING TANKER ASSIGNMENT PROBLEM

THESIS

RonJon Annaballi, Captain, USAF

AFIT/GOR/ENS/02-01

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

Report Documentation Page

Report Date	Report Type	Dates Covered (from... to)
11 Mar 2002	Final	July 2001 - March 2002

Title and Subtitle A Multiple Ant Colony Optimization Metahuristic for the Air Refueling Tanker Assignment Problem	**Contract Number**
	Grant Number
	Program Element Number
Author(s) RonJon Annaballi, Capt, USAF	**Project Number**
	Task Number
	Work Unit Number
Performing Organization Name(s) and Address(es) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 P Street, Bldg 640 WPAFB, OH 45433-7765	**Performing Organization Report Number** AFIT/GOR/ENS/02-01
Sponsoring/Monitoring Agency Name(s) and Address(es) Major Dave Ryer HQ AMC/XPY 402 Scott Drive, Unit 3L3 Scott AFB, IL 62225-5307 Major Juan Vasquez AFOSR 801 N. Randolph St., Room 933 Arlington, VA 22203-1977	**Sponsor/Monitor's Acronym(s)**
	Sponsor/Monitor's Report Number(s)

Distribution/Availability Statement
Approved for public release, distribution unlimited

Supplementary Notes
The original document contains color images.

Abstract
A key tenet to the Air Forces vision of Global Vigilance, Reach, and Power is the ability to project power via the use of aerial refueling. Scheduling of limited tanker resources is a major concern for Air Mobility Command (AMC). Currently the Combined Mating and Ranging Planning System (CMARPS) is used to plan aerial refueling operations, however due to the complex nature of the program and the length of time needed to run a scenario, the need for a simple tool that runs in much shorter time is desired. Ant colony algorithms are recently developed heuristics for finding solutions to difficult optimization problems based on simulation the foraging behavior of ant colonies. It is a distributive metaheuristic that combines an adaptive memory function with a local heuristic function to repeatedly construct possible solutions which can then be evaluated. Using multiple ant colony heuristics combined with a simple scheduling algorithm and modeling the Tanker Assignment Problem as a modified Multiple Depot Vehicle Routing Problem, an Excel based spreadsheet tool was developed which generates very good solutions in very short time.

Subject Terms Ant Colony optimization, Heuristics, Metaheuristics, tanker Scheduling, vehicle routing problem, multiple depot vehicle routing problem	
Report Classification unclassified	**Classification of this page** unclassified
Classification of Abstract unclassified	**Limitation of Abstract** UU
Number of Pages 98	

AFIT/GOR/ENS/02-01

A MULTIPLE ANT COLONY METAHEURISTIC FOR THE AIR REFUELING TANKER ASSIGNMENT PROBLEM

THESIS

Presented to the Faculty

Department of Operational Sciences

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Operations Research

RonJon Annaballi, BS

Captain, USAF

March 2002

AFIT/GOR/ENS/02-01

A MULTIPLE ANT COLONY METAHEURISTIC FOR THE AIR REFUELING TANKER ASSIGNMENT PROBLEM

RonJon Annaballi, BS
Captain, USAF

Approved:

/s/ 14 March 2002

James T. Moore, Ph.D. Date
Associate Professor
Department of Operational Sciences

/s/ 14 March 2002

Raymond R. Hill, Jr., Lt Col, USAF Date
Associate Professor
Department of Operational Sciences

Acknowledgments

I would like to thank the members of my thesis committee, Dr. Moore, COL Nanry, and Lt Col Hill for their patience with me while I changed the direction of my research three times. Their leadership, knowledge, and understanding were invaluable. Without them, I would not have been able to complete this thesis effort. Additional thanks go out to the rest of the AFIT faculty for providing me the tools necessary to finish my research, and ensure success in future analysis jobs. Thanks also go to Maj. Ryer at AMC for answering my questions about aerial refueling and to Capt Capehart, Lt Tekelioglu, and Capt Wiley for blazing the way in previous efforts.

A special thanks goes to my classmates. Although you tested my patience at times as Section Leader, this experience would have been unbearable without all of your support, stories, and countless games of Euchre. If you ever need a fourth, I'm in.

Finally, I want to thank my wife and son for their sacrifice during the past 18 months. Every day you teach me what life is really all about. I will never be able to truly express what you mean to me. I love you with all my heart.

Ron Annaballi

Table of Contents

List of Figures

List of Tables

Abstract

A key tenet to the Air Force's vision of Global Vigilance, Reach, and Power is the ability to project power via the use of aerial refueling. Scheduling of limited tanker resources is a major concern for Air Mobility Command (AMC). Currently the Combined Mating and Ranging Planning System (CMARPS) is used to plan aerial refueling operations, however due to the complex nature of the program and the length of time needed to run a scenario, the need for a simple tool that runs in much shorter time is desired.

Ant colony algorithms are recently developed heuristics for finding solutions to difficult optimization problems based on simulation the foraging behavior of ant colonies. They are distributive metaheuristics that combine an adaptive memory function with a local heuristic function to repeatedly construct possible solutions which can then be evaluated. Using multiple ant colony heuristics combined with a simple scheduling algorithm and modeling the Tanker Assignment Problem as a modified Multiple Depot Vehicle Routing Problem, an Excel based spreadsheet tool was developed which generates very good solutions in very short time.

A MULTIPLE ANT COLONY METAHEURISTIC FOR THE AIR REFUELING TANKER ASSIGNMENT PROBLEM

I. Introduction

Background

The United States Air Force prides itself in being able to rapidly project military power to any point in the world. The key to this vision of Global Vigilance, Reach and Power is aerial refueling or simply air refueling. Air refueling, the in-flight transfer of fuel from a tanker aircraft to a receiver aircraft, "supports the national military strategy across the spectrum of conflict, from peacetime operations for American global interest to major regional contingencies" (Iannuzzi, 1997b:15). In an age where the United States military continues to reduce its worldwide footprint, aerial refueling is sometimes the only way to meet current operational requirements and contingencies. It also provides the means to meet the Department of Defense's two war strategy as laid out in the 2001 Quadrennial Defense Review.

Air refueling was first demonstrated in 1923, when then Major Henry "Hap" Arnold performed the first in-flight hose contact between two De Havilland DH-4B aircraft (Iannuzzi, 1997a: 22). The Army Air Service continued to experiment with air refueling during the 1920s; however, it was not until 1929 when the true potential of in-flight refueling was demonstrated. The flight of the "Question Mark", a modified Fokker

C-2A, established new records for both air refueling and endurance. Using two modified Douglas C-1 biplanes as tankers, 143 contacts were made with the Question Mark allowing it to remain airborne for over 150 hours and an equivalent of 11,000 miles (Iannuzzi, 1997a: 22). Major Carl Spaatz commanded the flight and had the following to say afterwards:

> From a military standpoint the successful demonstration of air refueling means that bombing planes can now take off with heavy loads of bombs and little gasoline, aerial refuel, and continue to a more distant objective than would otherwise have been possible. (Iannuzzi, 1997a: 22)

After World War II, air refueling became an accepted strategy in the United States military. With the advent of strategic nuclear weapons and the requirement to reach any place in the world, air refueling was seen as the only method to meet this goal. The Strategic Air Command (SAC) was placed in charge of not only the entire fleet of nuclear weapon carrying aircraft but also the tanker forces to be used to get these aircraft to their worldwide targets. By 1957, SAC had 766 tankers, which comprised nearly 40 percent of its total aircraft inventory (Iannuzzi, 1997a: 23).

Today, practically all aircraft in the entire military fleet can be refueled while in the air. Aerial refueling operations fall under the control of the Air Mobility Command based at Scott Air Force Base, Illinois. The KC-135 Stratotanker and the KC-10 Extender are the primary air refueling tankers in the Air Force fleet.

The KC-135 entered the Air Force inventory in 1956 to extend the range of SAC's B-52 fleet (Capehart, 2000: 1). Basically a Boeing 707, the KC-135 is the workhorse of the fleet. It has a range of 1,500 miles with 150,000 pounds of transfer fuel with a ferry mission of 11,015 miles. (Air Force Magazine, 2000:143) As of January

2002, the Air Force had a total of 543 KC-135s in its Active Duty, Reserve and Guard fleet. (Airman, 2002:49)

The KC-10, a modified McDonnell Douglas DC-10, was introduced in the early 1980s to supplement the KC-135 fleet. It has a dual role in that it can also be used to carry cargo besides/instead of fuel. This ability to carry cargo eliminates the KC-10's reliance on forward basing and combined with its much larger fuel holding capability greatly reduces the number of KC-10s needed to support a deployment. Currently the Air Force has 50 KC-10s in its Active Duty fleet (Airman, 2002:49).

Table 1. Deliverable Fuel Capacity (radius of 2500NM)

Tanker type	Fuel (pounds)
KC-135A	63,000
KC-135R	94,500
KC-135E	75,600
KC-10	162,000

There are two types of missions that air refueling can perform. Aerial refueling can perform a "force-enhancement" or "deployment" role. This is the traditional role of tanker aircraft providing a global capability to U.S. aircraft. It makes possible intercontinental strategic airlift and strategic bombing without the use of forward basing. It allows the U.S. to quickly concentrate its power anywhere in the world. It can increase the payloads of aircraft by trading fuel load with cargo load. It allows short-range fighters to reach any part of the world (Iannuzzi, 1997a: 25).

The second role is that of a "force multiplier" in short-range theater operations. Aerial refueling allows strike aircraft to increase their combat radius, lengthen loiter time,

and carry heavier payloads while still returning safely to their operating bases (Iannuzzi, 1997a: 25). This is often known as an "employment" role.

Recent conflicts have proved the vital role aerial refueling plays in successful air operations. According to the *Gulf War Air Power Survey*, during Operation Desert Shield and Desert Storm, 400 tankers flew over 30,000 sorties and logged over 140,000 hours of flight time. Tanker aircraft off-loaded over 1.2 billion pounds of fuel to over 80,000 aircraft (Wiley, 2001: 2). During Operation Allied Force, nearly the entire fleet of tanker aircraft was used to transfer 356 million pounds of fuel to allied aircraft (Simpson, 2000: 10). More recently, in support of Operation Enduring Freedom, tankers played a critical role due to the lack of forward basing in the region surrounding Afghanistan. Seventy percent of the Air Force's active duty tanker fleet was tied up supporting the air campaign in Afghanistan (Newman, 2002: 57). This is a remarkable number seeing that the campaign rarely exceeded 100 strike sorties a day.

Problem Statement

Air Mobility Command's Tanker Airlift Control Center is responsible for planning and scheduling all tanker operations in support of air operations. During crisis situations, the number of aircraft being deployed and the distances required for travel can be immense. Determining the tanker support for these deployments and matching tanker aircraft with receiver groups can be a daunting process. AMC uses the combined Mating and Ranging Planning System to support their decision making process.

CMARPS is a computer simulation that analyzes, plans, and schedules deployment of tankers in support of military operations. It develops tanker/receiver

aircraft fueling schedules and flight plans taking into account aircraft limitations as well as Air Force rules and regulations. Due to the complexities of refueling scenarios, it can take up to two weeks for CMARPS to produce a schedule. This is inadequate to meet the many short notice contingencies that often arise in this constantly changing military environment. Additionally, CMARPS is not interoperable with other AMC airlift simulations. Since tanker and airlift missions are related and compete for limited airbase resources, some level of interaction is needed between the various simulation tools.

Research Objectives

Previous thesis efforts by Capehart (2000) and Tekelioglu (2001) have developed an Excel-based spreadsheet tool, called TAP, which can be used to solve the tanker assignment problem in much less time than CMARPS. They have interpreted the problem as both an assignment problem and a scheduling problem, and have used tabu search techniques to develop solutions.

The objective of this research is to improve TAP tool performance. This is done by adding increased functionality in the model to make it more operationally realistic. The approach to solving the problem is also changed as new heuristic methods are applied to the model.

While improving the TAP Tool, a strongly related task of the research is verification and validation (V&V) of the TAP Tool. V&V ensures that the model is producing correct results. It also highlights the strengths and weaknesses of the model. During development, verification is done by ensuring the model's algorithms produce the results that are expected, and that the internal logic of the program is valid. This is

accomplished through incremental code development and verifying the code works properly before continuing development. Validation is performed through comparison of model outputs for some small deployments with the results of other models.

Scope

TAP is a planning tool that models tanker deployment operations based on various factors. The tanker assignment problem is focused on answering the following questions:

1. Given receiver group deployment requirements, aircraft characteristics, system constraints, and defined tanker assets, how many tankers does it take to meet the receiver's refueling requirements?

2. Given the previous information, how quickly do receiver aircraft deploy and arrive at their desired location?

This problem involves assigning non-homogeneous vehicles, located at multiple locations, to meet the refueling needs of receiver groups. Time windows are also introduced to account for availability of aircraft over time as well as ensure receiver groups arrive when needed. In past efforts, this problem has been approached as both an assignment and scheduling problem. Tankers need to be assigned to receiver groups; therefore, it can be solved as an assignment problem. Meanwhile, because receiver groups have certain time availabilities, determining the time that tankers and receiver groups meet can be viewed as a scheduling problem. The tanker assignment problem can also be viewed as a vehicle routing problem and a set-covering problem.

Factors effecting this problem include the aircraft fuel capacities and burn rates, ground speed, true air speed, deployment distances, number of aircraft to be supported, time frames, locations of both tanker and receiver group origins and destinations, escort

requirement for receivers, and formation size. Other factors such as wind, altitude, and crew duty limitations are not explicitly modeled.

Contribution of Research

The goal of this research is to provide AMC with a quick running tanker assignment tool. This tool builds on past efforts by better representing the operational characteristics involved with aerial refueling. Because of the speed with which it executes, the tool can be used to perform more in-depth analysis of aerial refueling issues than with the current tools.

The tool uses an ant colony optimization algorithm to solve the tanker assignment problem. This is the first application of this heuristic technique to Air Force related research at AFIT. Upon searching the literature, it also appears to be the first application of ant colonies to solve a multiple depot vehicle routing problem.

Report Overview

This thesis is divided into the following five chapters: Introduction, Literature Review, Methodology, Results and Analysis, and Conclusions. A brief description of each follows.

Chapter 1: Introduction - This chapter discusses the background of the problem, defines the research focus and objectives, and illustrates the contribution of the research to the Air Force.

Chapter 2: Literature Review – This chapter begins with a discussion of the formulation of the tanker assignment problem. Next past efforts to solve the problem are discussed. Finally, heuristic methods and their applicability to the problem are evaluated.

Chapter 3: Methodology – The methodology chapter begins by describing the process for formulating the tanker assignment problem. The solution representation is defined. Next the heuristic to be used is developed, followed by its implementation as a spreadsheet model.

Chapter 4: Results and Analysis – This chapter presents the results produced by the spreadsheet model. Results are compared to those achieved by previous tools.

Chapter 5: *Conclusions and Recommendations* – The research results are reviewed and recommendations for further research are made.

II. Literature Review

This chapter begins with the introduction of the tanker assignment problem. Then possible ways to formulate the tanker assignment problem are illustrated. Next, recent approaches to scheduling aerial refueling are overviewed. Finally the field of heuristics is discussed including various techniques which could be employed.

Tanker Assignment Problem

The tanker assignment problem involves determining which tanker aircraft will service which receiver groups at what times. The characteristics of this problem are as follows:

- a set of receivers with their associated origin and destination bases along with known availability dates and required arrival times
- a set of tankers with their origin bases
- characteristics of both receiver and tanker aircraft (such as flight speed, fuel capacity, fuel burn rates)
- rules governing aerial refueling such as escort requirement

There are many ways to model the tanker assignment problem. A number of different interpretations are illustrated below.

Assignment Problem

The tanker assignment problem can be modeled as an assignment problem (AP), where you have *m* servers, *n* jobs, and a cost c_{ij} which is the cost of server *j* to perform job *i* (Winston, 1994: 373). Allowing the tankers to act as servers and the refueling points to act as jobs, the problem is to assign tankers to refueling points in such a way as to minimize the cost of completing all jobs, which in this case is the distance the tankers are required to travel.

The typical AP states that the number of servers must be greater than the number of jobs, and that each server can only perform one job (Winston, 1994: 373). To model the tanker assignment problem as an AP, the original problem must be relaxed to account for timing of events as well as allowing multiple tankers to service the same refuel point and allowing the same tanker to refuel multiple refuel points (Wiley, 2001: 18).

Set Covering Problem

The tanker assignment problem can also be modeled as a set covering problem. The set covering problem states that each member of a given set 1 (refueling points) must be covered by an acceptable member of set 2 (tankers). The goal is to minimize the number of elements in set 2 it takes to cover set 1 (Winston, 1994: 477). Once again if we relax the tanker assignment problem to account for multiple refuelings and time windows, then the set covering problem is a valid interpretation (Wiley, 2001: 17).

Job Shop Scheduling Problem

The tanker assignment problem can be modeled as a job shop scheduling problem (JSSP). The JSSP is made up of *n* jobs, composed of *m* ordered operations which must be assigned to a set of machines (Pinedo, 1995: 126). In this case the receiver groups act as

the jobs, the refueling points are the operations, and the tankers serve as the machines (Wiley, 2001: 16). The goal is to minimize the time it takes for all jobs (refueling points) to be processed on all the machines (tankers).

The problem with using JSSP is that the tanker assignment problem has multiple objectives. JSSP will minimize the time it takes for all jobs, but this may take an inordinate number of tankers. Thus using JSSP alone is not necessarily a good idea unless your only goal is to minimize the makespan, which may be valid in an emergency situation.

Vehicle Routing Problem

The tanker assignment problem can be viewed as a vehicle routing problem (VRP). The standard vehicle routing problem has one depot, which houses many vehicles, and *m* customers. The goal is to design vehicle routes such that each customer belongs to one route, that the demand along the route does not exceed the capacity of the vehicle, that each route starts and ends at the depot, and that the total duration of the routes are minimized (Cordeau, 1997: 105). In the tanker assignment problem, tanker aircraft play the role of the vehicle and the refueling points are the customers.

For this problem, a simple VRP is inadequate to effectively model the problem. Since many tanker bases can be used, a multi-depot vehicle routing problem (MDVRP) would more accurately model the problem. Additionally, since there are time windows in which aircraft can take off and land, a vehicle routing problem with time windows (VRPTW) would also be appropriate. The vehicle routing problem must still be relaxed to allow multiple tankers to service the same refueling point.

Tanker Scheduling Tools

A number of approaches have been applied to solving the tanker assignment problem with mixed results. Simulation and integer programming techniques were the first to be tried. More recently several heuristic methods based approaches have been implemented. The various methodologies are overviewed below.

Combined Mating and Ranging Planning System (CMARPS)

CMARPS is the current tool used by AMC to schedule air refueling. Originally developed in 1982, CMARPS is a deterministic simulation tool that aids in planning and scheduling deployment of tankers in support of global military operations. It assigns specific tankers to the required refueling points by attempting to minimize the number of tanker aircraft and the number of sorties flown for each mission.

The flight routes are developed considering restricted airspace, threat exposure, route deconfliction, and time over target requirements. This determines the fuel requirements. Finally, CMARPS assigns tankers considering tanker resources, tanker fuel consumption, air refuelable tankers, tanker reuse, and abort base requirements (Logicon, 1996).

CMARPS has a number of flaws. First because of the complexity of the tanker assignment problem, CMARPS can run extremely long, upwards of two weeks. It is also a very complex program which is difficult for even an experienced user to run. Finally, the extensive computing resources required "limits it efficiency, mobility, and versatility" (Wiley, 2001: 22)

Hostler's Air Refueling Tanker Scheduling Tool

Prior to 1987, SAC's scheduling system could only assign one tanker to refuel one receiver. As part of his 1987 AFIT thesis, Hostler developed a method to schedule SAC's fleet of air refueling tanker aircraft to perform more than one refueling per flight. He formulated the problem as a generalized assignment problem, which allowed each refueling request to be handled by more than one tanker as well as allowing one tanker to handle more than one request (Hostler, 1987: 21). Hostler used three objectives: maximizing the number of requests satisfied, maximizing the number of Category B requests satisfied, and minimizing the total flight time needed to fulfill all missions. Training requests were defined as either Category A, for normal air refueling training, or Category B, for training in support of formal courses, exercises, deployments, rotations, and tests.

A preemptive goal programming approach was used to solve the scheduling problem. This approach allowed the user to identify and prioritize the desired goals according to their importance. Each objective is then optimized based on its order of importance (Hostler, 1987: 21).

Hostler developed a preprocessor to put the data input from the tanker and receiver units in a proper format. This preprocessor determined all possible refueling combinations, sifting out any infeasible ones (Hostler, 1987: 59). Once in the proper format, the data was processed by an off-the-shelf integer programming package which solved the generalized assignment problem.

Hostler's work was a good first step in improving SAC's scheduling process. For simple training needs, the process would work fine. However for large deployments with

many tankers, this integer programming approach would ultimately run into the same problems of long run times as CMARPS.

Quick Look Tool for Tanker Deployment

Russina, Ruthsatz, and Russ developed an Excel based spreadsheet tool to evaluate tanker allocation for AMC in 1999. Written in Visual Basic for Applications (VBA) macros, the Quick Look Tool's goal was to determine the number of tankers needed to support a desired deployment as well as determine how quickly the deployment could be achieved (Wiley, 2001: 24). Tanker schedules were developed on a day-by-day basis.

Several simplifying assumptions were used. These include using only one type of tanker, allowing only one tanker to be assigned to a refueling point, knowing all refueling points ahead of time, and having all aircraft use constant flight speeds. Although this causes Quick Look to run very quickly, it decreases the accuracy of the schedules. Additionally, schedule precision is inadequate in that schedules are in terms of days, not the required hours or minutes (Tekelioglu, 2001: 10). Another drawback is the fact that schedules can be built for only one tanker base at a time.

TAP Tool

Two AFIT masters students, Capehart (2000) and Tekelioglu (2001) built upon the Quick Look Tool (QLT). Their model, called TAP, is an Excel based spreadsheet model coded in VBA. It is composed of multiple worksheets which provide input to the model regarding receiver groups and tanker bases. Capehart's model enhanced QLT by allowing the use of multiple tanker bases.

Both students interpreted the tanker assignment problem as an assignment problem (AP) with time windows. Due to the nature of this problem, a heuristic method was needed to obtain solutions in reasonable time (Tekelioglu, 2001: 10). Capehart chose to use a simple tabu search methodology (Glover and Laguna, 1997) with static short term memory to solve this problem. As problems increased in size and complexity, the time required to get good answers grew very rapidly. Also, the tabu search method employed used a static tabu tenure. Many runs had to be made to determine the proper tabu tenure for each scenario.

Additionally, Capehart made many simplifying assumptions. Each tanker was required to return to its origin based upon refueling, refueling occurred instantaneously at the refueling points, fighter aircraft are escorted over the entire flight, not just over large bodies of water, and wind and altitude effects on fuel burn and air speed are not modeled. Refueling points are generated ahead of time based on the maximum distance a receiver can travel on a full tank of fuel instead of trying to optimize the locations based on tanker base locations. Despite these facts, TAP was a huge improvement over past modeling efforts.

AMC improved the tool in the spring of 2000, by improving the graphical user interface. Separate worksheets were built for tanker and aircraft performance data. Their updates also gave the user the ability to easily make changes in the tanker aircraft information (Tekelioglu, 2001: 13). Changes were not made to the tabu search heuristic.

Tekelioglu built upon Capehart's work by using a primitive reactive tabu search method. This had the benefit of automatically adjusting the tabu tenure used by the heuristic based on the history of the search (Battitti and Tecchiolii, 1994: 128). This

eliminated the need to perform multiple runs to determine the optimal tenure value. Tekelioglu also performed some validation on Capehart's work and fixed some errors in calculating refueling points. Tekelioglu's model produced results similar to Capehart's work in similar periods of time.

Wiley's Group Theoretic Tabu Search Tool

Wiley attacked the tanker assignment problem as part of his doctoral research. He used a group theoretic tabu search methodology to solve the aerial fleet refueling problem (the deployment part of the tanker assignment problem). His JAVA based program (GTTS) has been shown to find much better results than those of the TAP tool. This is partially due to elimination of some of the assumptions made by TAP. Tankers are allowed to return to any tanker base instead of only their base of origin. Refueling points are defined so as to reduce the distance that tankers must travel to refuel receivers. Tankers can service more than just one refueling point (Wiley, 2001: 108).

Although GTTS provides very good results, there are some downsides. The main drawback is the amount of time it takes to develop a schedule. Runs take upwards of 30 minutes to produce good results (Wiley, 2001: 120). The TAP tool typically took half that time for the same scenario (Tekelioglu, 2001: 53). Another problem was the fact that Wiley's tool was not spreadsheet based, which created some difficulties with inputting receiver group information into the model. Future work is planned that will allow spreadsheet data to be modified as input files to the GTTS.

Barrel Allocator

The Barrel Allocator system (BA) was developed by Carnegie Mellon University (CMU), under a grant from the Defense Advanced Research Project Agency (DARPA).

It is a system for "solving complex transportation and logistics planning and scheduling applications" (Barrel, 2002). BA is an object oriented program that runs on a UNIX platform and is based on a scheduling framework (OZONE) developed at CMU. This framework is designed to insure rapid configurations of scheduling systems to incorporate the needed constraints, rules, and characteristics of a given problem set (Barrel, 2002). The framework was developed in Commonlisp (for scheduling) and JAVA (for the user interface).

The Barrel Allocator grew out of early attempts to schedule strategic airlift missions. Based on early successes, AMC and DARPA funded an initiative to extend the OZONE framework to cover all aspects of the airlift mission. This tool was to become part of the Consolidated Air Mobility Planning System (CAMPS) with the ultimate goal of being used operationally by the Tanker Airlift Control Center.

The Barrel Allocator performs a range of mission planning and scheduling tasks including generation of airlift mission schedules, generation of tanker missions, visualization and analysis of schedules, and mission merging. It is planned to support what-if analysis of options for mission planners (Barrel, 2002). Results so far have been impressive –"a 2 week interval of missions using approximately 1000 missions and 5000 flights took 45 seconds to schedule" (Barrel, 2002).

Currently the software does not run on a PC; however, it is envisioned that BA will eventually run on PC workstations. The main drawback is that tanker refueling is scheduled solely in support of airlift missions. The deployment of fighter and bomber aircraft is not supported.

Heuristics

Heuristics are defined as a method "for solving problems by an intuitive approach in which the structure of the problem can be interpreted and exploited intelligently to obtain a reasonable solution" (Silver, 1980: 153). Heuristics are often used to find near-optimal answers to complex problems in reasonable amounts of time. Zanakis and Evans present ten reasons for using heuristics. These reasons are presented below (Zanakis and Evans, 1981: 85-86):

1. Inexact or limited data used to estimate model parameters may contain errors much larger than that of a heuristic solution.

2. A simplified model may be used, which presents an inaccurate representation of the real problem. Therefore the exact answer to an approximate problem is found which may be worse than the approximate answer to an exact problem (which is what a heuristic finds).

3. A reliable exact method may not be available.

4. An exact method may be available, but it may be unattractive due to excessive computer requirements or computer run time.

5. Heuristics can often be used to develop starting solutions quickly to improve the performance of an optimization program.

6. The need for repeatedly finding problem solutions may result in significant computer savings.

7. A heuristic solution may be good enough. This is especially true when considering the time and resource savings.

8. The simple nature of heuristics are often easily understood by users which results in increased confidence in the solutions.

9. Heuristics can often be used to gain insight into complex problems, which aids in modeling the problem correctly.

10. Resource limitations such as time, money, and computer resources may force the use of a heuristic.

According to Silver, a good heuristic should have four properties. It should be solved with realistic computational effort. The solutions generated should be close to the optimal solution on average. The chances of a poor solution should be low. The heuristic should be simple for the user to understand (Silver, 1980: 155). Zanakis and Evans add that good heuristics should have reasonable core storage requirements. Heuristics should be robust in that the method should obtain good solutions for a wide variety of problems and should be insensitive to changes in the parameters. Heuristics should be able to handle multiple starting points and produce multiple solutions depending on the input parameters. Heuristics should have good stopping criteria which takes advantage of the search history, and the user should be able to interact with the method (Zanakis and Evans, 1981: 86-87).

Many heuristic methods have been developed. A few of the most recent developments in the field of heuristics are presented below.

Tabu Search

Tabu search was developed by Fred Glover in 1986. Tabu search allows the user to cross boundaries of feasibility or local optimality, which were usually treated as barriers (Glover and Laguna, 1997: 1) Tabu search explores the solution space by moving among the possible solutions. A move is made to some best solution in the neighborhood of the current solution. This move is not always an improving move. Recently visited solutions are made "tabu" which prevents these solutions from being revisited. This prevents cycling from occurring and helps the heuristic avoid getting stuck at a local optimality. The tabu nature of previous solutions also helps direct the search to visit previously unexplored regions of the solution space.

Tabu search relies heavily on both short term and long term memory to control the search. A tabu list, which consists of recently visited solutions, helps determine which direction the search will take. The size of the list determines how long a solution will remain tabu. Two key elements of tabu search are intensification and diversification. Intensification intensifies the search in a region which has previously produced good solutions. Diversification is used to move the search to an area of the solution space previously unexplored. By using intensification and diversification, tabu search can explore a fraction of the solution space in a reasonably short period of time and find very good solutions.

Tabu search has been used quite extensively on a number of different problem types. In his book, Glover details many applications of tabu search. These include planning and scheduling problems, vehicle routing problems, assignment problems, integer programming, and many others (Glover and Laguna, 1997: 267-303)

Ant Colonies

Ant colony optimization metaheuristics were developed in the early 1990s by Marco Dorigo. These ant colony approaches are based on the observation that real ants have an uncanny ability to find the optimal path to a food source (Bullnheimer, 1997). It was found that ants leave a trail of pheromone behind them. The amount of pheromone left behind is dependent on the length of the path taken as well as the quality of the food source (Bullnheimer, 1999: 322). Later ants are influenced by this pheromone, taking the path with the heaviest amount of pheromone. As time goes by, the optimal path will emerge as all ants choose to take the path with the heaviest amount of pheromone.

The ant colony heuristic imitates this behavior of ants. Artificial ants are used to explore the solution space. They make their moves based on some defined probabilistic function of the local neighborhood. Through these moves, the ants build possible solutions to the problem at hand. After a solution has been built, the ants lay down a weighting factor (pheromone) on the path that the ant took based on the quality of the solution found (Dorigo, 1999). The pheromone trail will be used by ants in future iterations to direct the search.

The ant colony metaheuristic has been used quite successfully for many shortest path type problems. Dorigo first applied it to the Traveling Salesman Problem. Later work has been performed on the quadratic assignment problem, vehicle routing problems, and network routing problems, to name just a few applications (Corne, 1999:30).

Scheduling Theory

This section introduces concepts from scheduling theory, especially as it relates to project management and the targeting process. "Scheduling concerns the allocation of limited resources to tasks over time. It is a decision-making process that has as a goal the optimization of one or more objectives" (Pinedo, p.1, 1995). The scheduling process exists in virtually all settings, whether in commercial or military environments. It is especially important in manufacturing arenas. In the military, scheduling is widely used in settings such as weapon system development (acquisition) and flight scheduling (Calhoun, 2000: 22).

A schedule is made up of resources, tasks, and objectives. *Resources* may be machines in a workshop, runways at an airport, aircraft in a squadron, tanker

aircraft available and so forth. *Tasks* may be operations on an assembly line, stages in a construction project, attacking targets on a Target Nomination List (TNL), or refueling points, for example (Calhoun, 2000: 23). *Objectives* include the minimization of the completion time of the last job (makespan), minimization of the maximum tardiness (worst violation of the due dates), and minimization of the total number of late tasks, to name a few (Pinedo, 1995: 1).

Parallel Machine Models

A machine can be thought of as a finite resource required for completing a task, such as a drill press in a job shop, cashiers in a checkout line, or tanker aircraft in theater. The simplest model is that of the single machine, and many algorithms have been developed to solve these simple models. However, in most real-world settings, the occurrence of parallel machine models is more common (Pinedo, 1995: 61). Heuristics have been developed which can solve these more complex problems. When parallel machines are present, job *j* requiring processing on a single machine, may be processed on any of the machines in the shop (Calhoun, 2000: 24). For example, if the job is to refuel an aircraft, it can be accomplished by any of the tanker aircraft in the area. Parallel machines can be identical (jobs are processed at the same rate regardless of machine chosen) or unrelated (process time depends on which machine is selected).

One of the most common objectives of scheduling problems is that of minimizing the makespan, or completion time of the last job. Often schedulers must deal with balancing the load across the machines in parallel; by minimizing the makespan, a good balance is ensured (Pinedo, 1995: 61). Scheduling parallel

machines may be considered a two-step process. First, determine which jobs should be allocated to which machine. Second, determine the sequence of jobs on each machine, subject to any precedence constraints (Pinedo, 1995: 62).

Precedence Constraints

Precedence constraints define timing requirements between activity pairs within projects. The most common type of precedence constraints are of the finish-start variety and are used to specify that a predecessor activity must end before its successor activity More complex activity timing requirements can be expressed by generalized precedence constraints which dictate a minimum lag time between an endpoint of a predecessor activity and an endpoint of a successor activity. (Calhoun: 2000:25)

Precedence constraints among activities in a project may make the project hard to explain verbally or via a mathematical model. Therefore graphical representations of precedence constraints, such as *activity on the arc* (AOA), are frequently used. In an AOA representation, a node designates an event in the network and an arc is directed from node *i* to node *j* if and only if event *i* must be completed before the activity leaving node *j* can begin. The duration of the activity is indicated on the arc. The boldface arrows denote the *critical path* in Figure 1. The AOA model is usually associated with *critical path method* analysis (see next section) and is the basis for most computer implementations (Calhoun, 2000: 27).

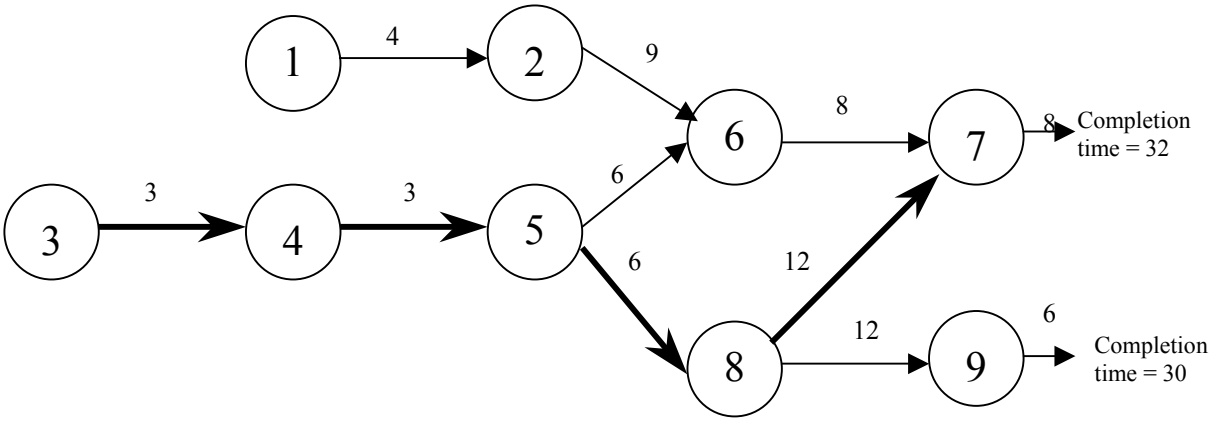

Figure 1. AOA Network Representation

Precedence constraints may also be represented by an *activity on the node* (AON) network. In an AON representation, a node designates an activity in the network and may display information about the activity such as duration, early start (ES), early finish (EF), late start (LS), and late finish (LF). Arcs depict precedence relationships. A typical node in an AON network would be:

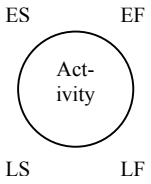

Figure 2. AON Node Representation

The advantage of using an AON network is that the calculations for project completion times may be displayed directly on the nodes by using the Critical Path Method forwards and then backwards (Calhoun, 2000:28).

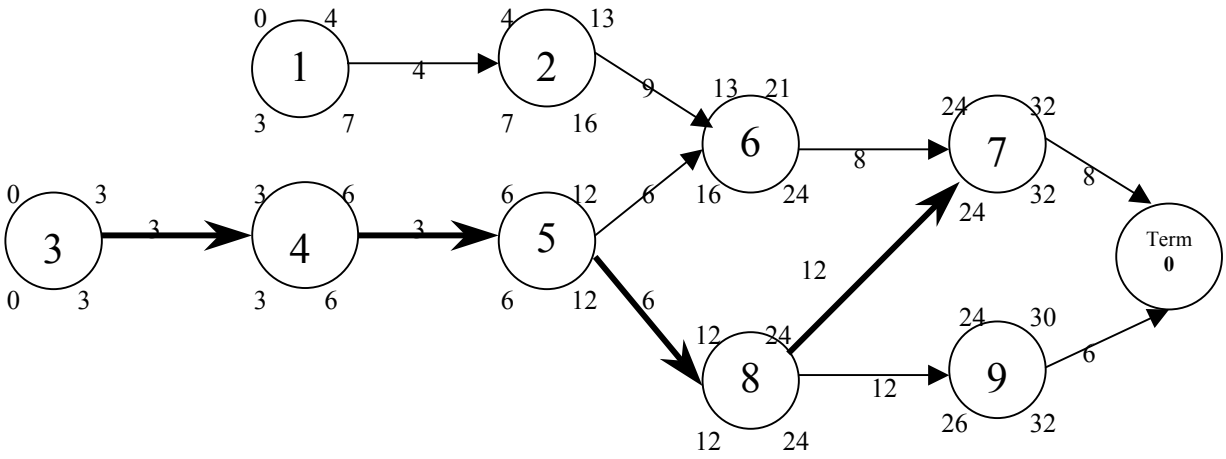

Figure 3. AON Network Representation

Critical Path Method

The Critical Path Method (CPM) for project scheduling uses either an AON or an AOA network for graphically portraying the relationships between the tasks and milestones in a project. When the number of resources are unlimited, or at least as large as the number of jobs, the CPM technique yields a schedule with an optimal makespan. Pinedo defines the CPM algortithm as: (Pinedo, 1995: 65)

1. Schedule the jobs one at a time starting at time 0.
2. Whenever a job has been completed, start all jobs for which all predecessors have been completed (i.e. all schedulable jobs).

The *critical path* is the set of jobs that cannot be postponed without delaying the earliest finish of the final schedule. These jobs are called critical jobs while jobs not on the critical path are called slack jobs (Pinedo, 1995: 65). The length of the critical path (or any other path) is equal to the sum of the durations of every activity on that path. If the earliest finish equals the due date, then duration of the entire project is equal to the length of the critical path (Pinedo, 1995: 65).

Summary

This chapter began by providing background on the tanker assignment problem. A number of possible ways to interpret the problem are overviewed. This is followed by a look at previous attempts to schedule tanker assets. An overview of heuristics is provided focusing on two more recent methods: tabu search and ant colony optimization. Finally, some key elements of scheduling theory and project management were outlined including network diagrams and the

critical path method. The next chapter applies some of these techniques to the Tanker Assignment Problem.

III. Methodology

Introduction

This chapter details how the topics and methods from Chapter 2 were applied to the tanker assignment problem. An explanation of the vehicle routing problem (VRP) formulation used to solve the problem is provided. The methods used to convert the tanker assignment problem into a VRP are discussed. The chapter finishes with a detailed look at the ant colony heuristic and scheduling methodology developed to schedule tanker assets.

Vehicle Routing Problem

The VRP can be represented by a complete weighted graph G = (V, A, d) where V is a set of vertices and A is a set of arcs and d is the distance of the arcs. The vertices consist of the depots and the customers and the nonnegative weights d_{ij} associated with each arc represent the distance between v_i and v_j. Each vehicle has capacity and maximum route delivery constraints. The goal is to find a route where:

1. each customer is visited exactly once by exactly one vehicle
2. all vehicle routes start and end at the same depot
3. the total demand of each route much not exceed the capacity of the vehicle
4. the total route length cannot exceed a given bound

To form the VRP, let *nc* be the number of customers. Let *v* index *nv* vehicles and let x_{ij}^{v} equal one if vehicle *v* goes from *i* to *j* and zero otherwise. Let c_{ij} be the cost of traveling from customer *i* to customer *j*. Let t_{ij} be the time it takes to travel from *i* to *j*. Let s_j be the service time at *j*. Let *R* be the limiting range in time of the vehicle. To account for the vehicles' capacities to deliver products, let d_i be the demand of customer *i* and *D* be the capacity of each of the vehicles. This assumes the vehicles are identical, or homogeneous.

The tanker assignment problem can be viewed as a specific form of the vehicle routing problem. Since the Air Force uses multiple tanker platforms, the VRP has heterogeneous vehicles. To account for heterogeneous vehicles, many of the variables need to be indexed by the vehicle *v*: c_{ij} becomes c_{ij}^{v}, s_j becomes s_j^{v}, *R* becomes R^v, and *D* becomes D^v.

Additionally, since receiver groups have defined departure and delivery dates, there are time windows associated with the refueling of aircraft; thus, we have a vehicle routing problem with time windows (VRPTW). Generally, the tanker may arrive early but must then wait until the beginning of the time window, which is not desirable. Let e_i be the earliest arrival time, let l_i be the latest arrival time, and let s_i be the service time for customer *i*. Let t_{ij} be the travel time between customers *i* and *j*. Let A_i and T_i be the time a vehicle arrives at customer *i* and the time it begins servicing customer *i*, respectively. Let W_i be the time spent waiting for service to begin at customer *i*. Finally, since there are multiple tanker bases around the world, there are multiple depots available in the vehicle routing problem.

The objective is:

$$\min \; Z = \sum_{v=1}^{nv} \sum_{i=1}^{nc} \sum_{j=1}^{nc} x_{ij}^{v} c_{ij}^{v} \tag{1}$$

{Minimize total cost}

subject to

$$\sum_{v=1}^{nv} \sum_{i=0}^{nc} x_{ij}^{v} = 1 \quad \forall\, j = 1..nc \tag{2}$$

$$\sum_{v=1}^{nv} \sum_{j=1}^{nc} x_{ij}^{v} = 1 \quad \forall\, i = 0..nc \tag{3}$$

$$\sum_{i=0}^{nc} \sum_{j=1}^{nc} x_{ij}^{v} t_{ij}^{v} + \sum_{i=0}^{nc} \sum_{j=1}^{nc} x_{ij}^{v} s_{j}^{v} + \sum_{i=0}^{nc} \sum_{j=1}^{nc} x_{ij}^{v} W_{j} \le R^{v} \quad \forall\, v = 1..nv \tag{4}$$

$$x_{ij}^{v} \in \{0,1\}$$

$$i = 0..nc, \quad j = 1..nc, \quad v = 1..nv \tag{5}$$

$$\sum_{i=1}^{nc} \sum_{\substack{j=1 \\ j \neq i}}^{nc} x_{ij}^{v} \le nc - 1 \quad \forall\, v = 1..nv \tag{6}$$

$$\sum_{i \in V^{v}} d_{i} \le D^{v} \quad \forall\, v = 1..nv \tag{7}$$

$$\text{if } x_{ij}^{v} = 1 \text{ then } T_{i} + s_{j}^{v} + t_{ij}^{v} + W_{j} \le T_{j} \tag{8}$$

$$e_{i} \le T_{i} \le l_{i} \quad \forall\, i = 1..nc \tag{9}$$

Constraint (2) specifies that only one vehicle can visit each customer. Constraint (3) ensures that only one vehicle can leave each customer. Constraint (4) limits the time that a vehicle can be made available. Constraint (5) defines the domain based on the size of the problem. Constraint (6) is a standard subtour breaking constraint. Constraint (7) limits the capacity a vehicle can carry, while constraint (8) puts a time limit on the routes. Finally, constraint (9) defines the time window for each customer.

Tanker Assignment Problem as a VRP

A VRP tries to determine the best routes for a number of vehicles to make to service a given number of customers. To interpret the tanker assignment problem as a vehicle routing problem, the tankers are the vehicles, the tanker bases are the depots, and the refueling points are the customers.

The key to this is determining where the refueling points should be located. I initially determine my refueling points like Capehart (2000). First, find the great circle distance from the receiver group's base of origin to its destination. Then determine the distance the receiver group can travel before it reaches its fuel reserve limit. This distance is determined simply by using fuel flow; it does not incorporate any sophisticated fuel burn equations which account for other problem specific factors like wind and cargo. For the initial leg of the flight, the amount of fuel burned during the climb is also accounted for. The offload at each refueling point is equal to the amount of fuel needed to fill each aircraft to its maximum fuel load, except for the final refueling point where only the necessary amount of fuel needed to get the receiver group to the destination base is offloaded.

A key aspect of the vehicle routing problem is that each customer is visited only once by exactly one aircraft. The offload required at a refueling point may exceed the offload capacity of a single tanker. This is especially true when considering long distance sorties for bombers or when using large receiver groups. To combat this, I determine the minimum number of tankers required to refuel each refueling point and then split the large refueling point into a number of smaller refueling points, each with equal offload amounts.

Each customer has a set service time that it takes a vehicle to perform a task. For simplicity sake, it is assumed in this problem that tankers can instantaneously pass the desired fuel offload to the receiver groups, resulting in a service time of zero. Additionally, although the VRPTW allows a vehicle to wait at a customer if it arrives early, this is not accounted for in this program. Thus, waiting times are eliminated from the schedule.

A final consideration with the tanker assignment problem is the fact that fighter aircraft require tanker escort over large bodies of water. To deal with this matter, I create an additional refueling point with zero offload to match the coordinates of each distinct refueling point. I require that a single tanker must be assigned to each pair of zero offload refueling points in a receiver group. This is discussed in later sections.

Ant Colony Metaheuristic

For the purpose of this research, an ant colony metaheuristic was chosen. Artificial ants are defined which choose which customers to add to the route in a step-by-step process. When no additional customers can be added without exceeding either

capacity or route length constraints, the ant returns to the depot. Pheromone (τ) is then deposited along the route taken.

The ant determines its path based on two factors: (1) the closeness of the next customer and (2) the pheromone trail. Closeness, n_{ij}, is a static heuristic variable. It is simply the inverse of the distance between points *i* and *j*, and specifies how promising the choice of that customer is. The pheromone trail, τ_{ij}, is a dynamic variable which signifies how good the choice of that customer was. It is computed by adding the closeness values of all arcs used in the route. The following proportional rule determines which customer will be visited next by the ant where Ω is defined as the neighborhood of the current customer.

$$p_{ij} = \frac{[\tau_{ij}]^{\alpha}[n_{ij}]^{\beta}}{\sum_{h \in \Omega}[\tau_{ih}]^{\alpha}[n_{ih}]^{\beta}} \quad \text{if } v_j \in \Omega \tag{10}$$

The probability distribution is biased by the parameters α and β which determine the influence that the pheromone trail and the local heuristic have on the selection process. If α is equal to zero, then the function acts like a greedy algorithm simply choosing the closest customer. If β is equal to zero, then only the pheromone trails are used to select customers. This would lead to all ants choosing the same tour, which in general would lead to a suboptimal solution. It has been experimentally found by Dorigo that parameter values α =1 and β = 5 were good (Corne, 1999: 22).

A key aspect of pheromone trails is the concept of evaporation. To prevent ants from converging too quickly to a suboptimal path, pheromone trails are allowed to

"evaporate" in order to encourage exploration of different paths during the search process (Bonabeau, 2000: 74). This is done by decreasing the current pheromone level of an arc by some exponential variable, ρ, before adding the new pheromone for the new route. For instance when an ant moves from city *i* to city *j*, the pheromone along the arc *ij* changes based on the following formula where :

$$\tau_{ij}^{new} = \rho\tau_{ij}^{old} + \sum_{k=1}^{m}\tau_{ij}^{k} \quad (11)$$

Dorigo experimentally found that a parameter setting of ρ = 0.5 was good (Corne, 1999, 22).

The number of artificial ants used in the heuristic is set equal to the number of customers in the VRP and each ant starts its tour from a separate customer. Initially we place one ant at each customer. Once this is done, we follow a two step process of first building vehicle routes and then updating the pheromone trail. This process is continued for a given number of iterations.

Because the tanker assignment problem more often than not includes multiple tanker base locations, this results in a multiple depot vehicle routing problem with each tanker base acting as a depot. To solve this problem, I choose to break the problem up into multiple vehicle routing problems with each tanker base acting as a depot in its own problem. Using the ant colony metaheuristic described prior, the VRP is solved for each tanker base.

Once these individual problems are solved, the pheromone trails constructed from each problem are then added together to construct a pheromone trail matrix for the entire

problem. The ant colony heuristic is then used a final time to construct entire solutions to the given problem.

Scheduling

The ant colony metaheuristic is used to construct possible solutions. Once these solutions have been produced, it must be determined whether or not these solutions are feasible in regards to the timing constraints. Using an AON network structure, it is possible to determine the earliest start and latest start for each refueling point. It is also possible to determine the earliest finish and latest finish for the refueling points; however since it is assumed that refueling operations are instantaneous as far as offload goes, these finish times are the same as the start times.

In terms of scheduling theory, we have a job shop where the receiver groups can be interpreted as jobs, the refueling points can be seen as operations per job, the tankers are the machines and the processing times at each machine are equal to zero. We also have precedence constraints on the jobs where refueling point 1 for receiver group 1 must be scheduled before refueling point 2 of receiver group 1, and so on. If we assume that on average there are q refueling points per receiver group, m tankers, and n receiver groups, then there are as many as $[(nq/m)!]^m$ schedules.

There are numerous dispatch heuristic methods which can be used to build a schedule one step at a time. These methods can be very inexpensive to find solutions to complex problem and can often be used repeatedly to develop more sophisticated search heuristics. Some of the more popular dispatch heuristics include the Earliest Due Date

(EDD), the Weighted Shortest Processing Time (WSPT), First Come First Served (FCFS) and the Minimum Slack Time (MST) (Morton, 1993: 374).

To build the schedules for the given routes, I chose to use an EDD dispatch heuristic. Taking the routes generated by the Ant Colony heuristic, I parse out the routes to determine which refueling points are assigned to which tanker bases. The next step is to determine how long it takes for the assigned tanker to reach the first refueling point for each receiver group. This is important, because even though we already have a time window associated with each refueling point, we do not want our receiver groups to wait for our tankers to arrive and thus this calculation determines the true earliest start time a refueling point can be hit by the receiver group. Once we set the refueling time for the first refueling point in the receiver group, then the refueling point for the rest of the refueling points for the receiver group can be calculated. Once again we assume no waiting time for the receiver groups.

Once we have the updated time windows developed for the first refueling point of each receiver group, we can schedule the tankers based on a least slack time rule. We determine which receiver group has the least slack built in their schedule and start scheduling this receiver group first. If a conflict occurs, such as two tankers from the same base are required at different refueling points at the same time, then we can either assign another tanker from that same base to service the second refueling point or we can delay the receiver group's start time such that it can be serviced by the first tanker at a later time, as long as the receiver group still arrives at its destination point on time. This process is continued for all refueling points.

Visual Basic With Applications

Visual Basic with Applications (VBA) is Microsoft's common scripting language. It is included in all Microsoft Office applications and is also part of many applications from other vendors. The key to the VBA language is that you can create structured programs directly within Excel.

There are a number of reason why it was decided to develop this program in VBA. First, AMC desired a tool that was easy to use. One of the weaknesses of their current tool, CMARPS, was the difficulty associated with learning and using the program. Since most Air Force members have a familiarity with Excel and Excel based spreadsheets, it was the obvious choice as the platform for this tool. Since VBA is the built in scripting language for Excel, it too was an obvious choice. Furthermore since previous efforts were also Excel based spreadsheets coded in VBA, these presented a good starting reference for this work. Additionally, Excel's built in functions provides a good means to analyze results.

Measurement of Results

Ideally, it would be nice to compare the results obtained with this tool to those developed by CMARPS. However, due to the time needed to learn how to efficiently use CMARPS and the lack of trained personnel at AMC, it was impossible to obtain CMARPS solutions for the desired scenarios. Results from this model are compared to those obtained using the previous TAP tool developed by Capehart and Tekelioglu. Some very simple CMARPS scenarios for a single aircraft and a small receiver

deployment have been provided by AMC. The results obtained by CMARPS are checked against this model. The next chapter reports the results of model testing.

IV. Results and Analysis

An Excel-based tool was developed to input a number of receiver group requests and output a mission plan consisting of tanker assignments to refueling points. Two sample deployments were provided by AMC for testing this new tool. In addition, AMC provided the bed-down locations of KC-135s around the globe.

Since this tool considers all tankers within range of refueling points, it is necessary to decrease the actual numbers and locations of tankers in order to increase the computational efficiency of the tool. For example, there are several Air National Guard and Air Force Reserve bases in the U.S. that contain tanker aircraft. Many of these bases are within range of the early refueling points of a deployment. This large number of available tankers greatly increases the complexity of the problem, which in turn increases the time needed to obtain a solution. Therefore, we first run the program with tankers located at active airbases. If the tool returns a solution with no tanker assigned to a refueling location, we can then go back and start placing tankers at other bases originally in the list of bed-down locations so that there is a tanker capable of satisfying the receiver group's requirements at that refueling point.

Southeast Asia Deployment

The first deployment tested involves receiver groups departing the continental U.S. and arriving in Southeast Asia. Table 2 provides a list of the 11 receiver groups shown in Figure 4.

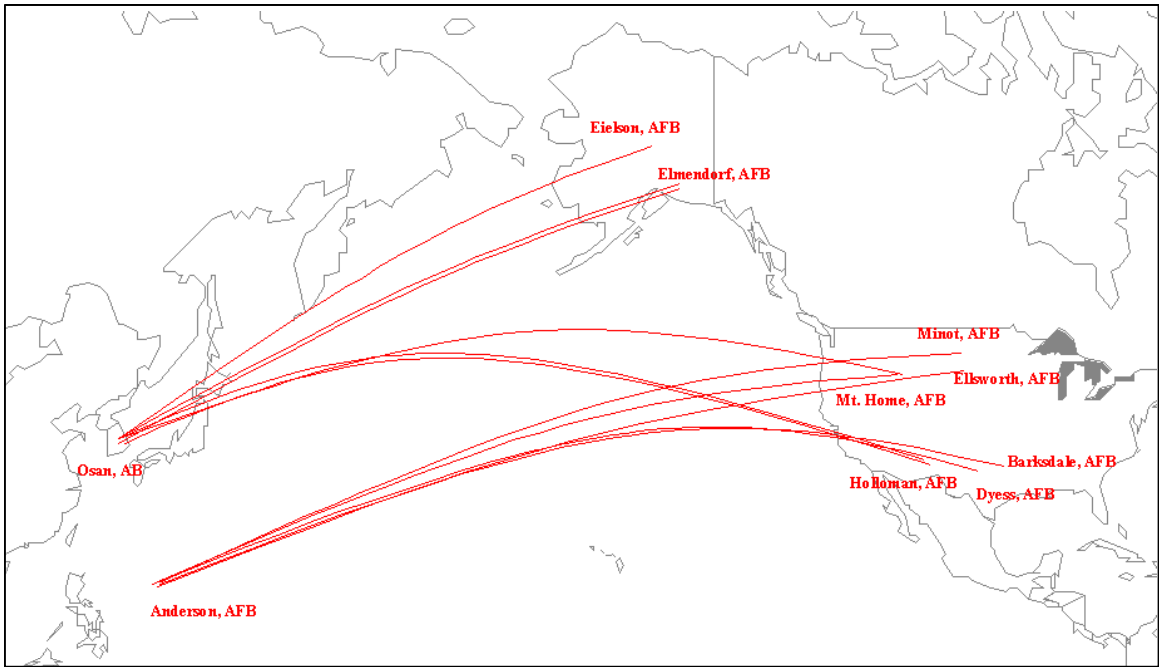

Figure 4. Southeast Asia Deployment

The tanker bases activated for this deployment include McConnell, Mountain Home, Grand Forks, Fairchild, Kadena, and Eielson, with 15 KC-135 tankers located at each base. Tankers located at these bases are capable of satisfying all the receiver groups' fuel requirements during the deployment. Although the model supports the use of waypoints, none are used for this deployment. A waypoint is a location the receiver group must first reach before heading towards their destination base. Our code allows the user to input one waypoint for each receiver group.

Table 2. Receiver Groups for Southeast Asia Deployment

Aircraft Type	Number of aircraft in group	Origin	Destination	ALD	RDD
F117	2	Holloman	Osan	1	5
F15	6	Mountain Home	Osan	1	5
F15	6	Elmendorf	Osan	1	5
F16	6	Eielson	Osan	1	5
A/OA10	6	Eielson	Osan	1	5
B1	1	Mountain Home	Andersen	1	5
B1	1	Ellsworth	Andersen	1	5
B1	1	Dyess	Andersen	1	5
B52	1	Barksdale	Andersen	1	5
B52	1	Minot	Andersen	1	5
F117	2	Holloman	Osan	1	5

This initial mission plan needed a total of 27 tankers. After running the scheduling procedure, a final schedule requiring 15 tankers was found. The best solution found using the model required 91,266 tanker miles and was obtained in under 3 minutes on a 700 mHz Pentium III.

The initial mission plan generated by the previous TAP models by Capehart and Tekelioglu required 28 tankers. After running their tabu search methodology, a final schedule was constructed requiring 13 different tankers. The best solution required 120,461 tanker miles and was obtained in 15 minutes on a 700 mHz Pentium III.

Table 3. Best Schedule for Southeast Asia Deployment

Tanker / Tail Number	Tanker Takeoff	Tanker Land	RP #1	RP #2	RP #3	RP #4	RP #5
MOUNTAIN HOME AFB / 9	1.657064	7.701881	1	2	3	4	
EIELSON AFB / 10	3.204402	14.80486	4	5	6	8	
KADENA AB /	81.80084	90.14088	8	7	8	9	10
EIELSON AFB / 13	1.871436	4.350696	11	12	13		
EIELSON AFB /	84.50398	93.2129	13	16	14	16	
EIELSON AFB /	85.96849	93.2129	16	15	16		
EIELSON AFB /	85.96849	100.9233	16	18			
KADENA AB /	91.76464	95.11093	18	17	18		
EIELSON AFB / 14	2.406679	8.972357	19	21			
EIELSON AFB /	82.36301	88.92869	21	20	21		
EIELSON AFB /	82.36301	96.7238	21	23			
KADENA AB /	87.59273	91.39314	23	22	23		
EIELSON AFB / 15	0	5.020739	24	25			
EIELSON AFB /	74.18486	85.199	25	26	27		
EIELSON AFB / 12	1.665528	8.944501	28	30			
EIELSON AFB /	85.32761	92.60659	30	29	30		
EIELSON AFB /	85.32761	102.2807	30	31	32		
KADENA AB / 11	6.940407	13.89736	33				
KADENA AB / 8	3.672733	14.85161	34				
KADENA AB / 4	5.202388	15.63538	35				
KADENA AB / 5	5.202388	15.63538	36				
EIELSON AFB / 1	3.358625	14.76722	37				
EIELSON AFB / 2	3.358625	14.76722	38				
EIELSON AFB / 3	2.106071	15.49878	39				
MOUNTAIN HOME AFB / 6	3.272273	9.31709	40	41	42	43	
EIELSON AFB / 7	4.819611	16.42007	43	44	45	47	
KADENA AB /	62.35475	70.69479	47	46	47	48	49
			Tankers Used: 15				
			Total Tanker Distance: 91266.43 miles				

CMARPS vs. TAP Tool

In order to compare the effort required by CMARPS to that of the TAP tool, we use two simple mission plans. The first mission plan involves a single B-52 receiver group scheduled to fly from the U.S. to Souda. We look at both one-way and round-trip

flights. AMC is interested in the comparison of computation time and total offload required by the B-52 for this mission. Table 4 displays the comparison of the two tools.

Table 4. Tool comparison for B-52 mission

	One-way (KBAD→LGSA)			Round-trip (KBAD→LGSA→KBAD)		
	Setup Time (min)	Runtime (min)	Offload (K lbs.)	Setup Time (min)	Runtime (min)	Offload (K lbs.)
CMARPS	20	5	0	20	5	250
TAP (Capehart)	< 1	0.05	19.2	< 1	0.06	240
TAP Tool	< 1	< 0.01	19.2	< 1	< 0.01	240

The difference in offload required for the one-way flight is probably due to the climb-fuel required by the B-52. The B-52 can fly 5663 miles on a full tank of fuel. The total flight distance for this mission is 5428 miles. However, after the B-52 uses 19,200 lbs. of fuel to climb to altitude, it can only travel 5213 miles with the remaining fuel on board. Depending on the distance the B-52 travels during its climb to altitude, it may have less than 5213 miles remaining in the flight. This is obviously the case in the CMARPS run, since there was no requirement for a refueling point. In our tool, we assume that every aircraft type flies 100 miles during the climb. In this case, the B-52 has 5328 miles remaining when it reaches altitude, which is greater than the number of miles the B-52 can fly with its remaining fuel. The actual number of miles traveled during an aircraft climb to altitude is dependent on the winds and the target altitude.

The second small mission for comparison involves three fighter receiver groups. The groups consist of 6 F-15s, 6 F-16s, and 2 A-10s. Their destination base is Lajes AB, while their origin bases are St. Louis-Lambert International, Duluth, and Hurlburt Field, respectively. Again, we are interested in the time to run this mission and the total offload

required for the receiver groups. Table 5 compares the results of CMARPS with that of the TAP tool.

Table 5. Tool comparison for fighter missions

			Total Offload Required		
	Set-up Time (min)	**Run Time (min)**	**F-16**	**F-15**	**A-10**
CMARPS	20 - 26	9 - 15	105.1	75.5	28.2
TAP Tool	1	< 0.01	125.4	80.4	38.5

The total offload required for each aircraft provided by the TAP tool is reasonably close to those provided by CMARPS, according to Maj. Dave Ryer, AMC. The computation time for the TAP tool is significantly less than that of CMARPS. This computational time benefit is of major interest to AMC.

Model Improvements

A number of improvements have been made to this model. First is the ability for tanker aircraft to support multiple refueling points. The model of Capehart and Tekelioglu only allowed a tanker to refuel one refueling point. This is not operationally realistic and was extremely inefficient. Besides increasing realism, allowing tankers to refuel multiple refueling points decreases the number of tankers needed to support a deployment, as well as decreasing the total flight distance needed.

Another improvement was the ability to model several types of tankers. Previous TAP models only modeled the KC-135R aircraft. This version models the Air Force tanker fleet. In addition, it allows the user to "invent" new tanker aircraft. Thus this tool can be used to test new aircraft designs and analyze their impacts on current operations.

A graphical user interface (GUI) was added to the model which allows the user to access the various worksheets by way of call buttons. Each of the input screens also have call buttons which allow the user to search various databases for information necessary for that input screen. For instance, when defining the receiver group flight paths, the user is allowed to access the worldwide airfield directory to double check airfield code names. Finally each input screen has error checking which insures that all of the information entered is of the proper format, that the information is correct , i.e. that the airfield code name actually exists, and that all required information is entered. If errors are found, the user is directed as to how to fix the input. This insures that the model cannot be run without having all information required, thus reducing the possibility of the model crashing.

Finally, this tool fixes problems with previous TAP tools. Although Capehart and Tekelioglu allowed waypoints to be entered, they never tested this ability. Thus they did not see that the way their models calculated the effect of waypoints was incorrect. In fact, waypoints were not considered when generating refueling points due to an error in the way the inputted waypoints coordinates were defined in the model code. This problem has been fixed in this model.

V. Conclusions

Review

Scheduling of aerial refueling operations is a complex task. Because of the nature of the problem, the time necessary to solve even small problems to optimality can be great. For that reason, heuristic methods can be used to provide very good solutions to the Tanker Assignment Problem in reasonable time.

This research built off the previous work by Capehart and Tekelioglu, who developed spreadsheet models using tabu search metaheuristics to schedule tanker refueling operations. It was found that these models did provide very good solutions in much faster time than the model currently used at AMC.

For this research, an ant colony metaheuristic was developed which was used in the spreadsheet model to generate tanker refueling results. This ant colony routing heuristic was combined with a simple least slack time algorithm to find solutions to the problem. It was found, through comparison to the previously developed spreadsheet models, that this ant colony based model found much better solutions in terms of minimizing the amount of distance tankers are required to travel. It also did this in much smaller periods of time.

Since almost every computer in the Air Force contains Microsoft Office with Excel, this tool is very portable. Also, most users are familiar with how to enter and manipulate data within Excel spreadsheets. This increases the usability of the tool and allows new personnel to use the tool with minimal training.

Recommendations

This model provides very good solutions very quickly. However, in order to do this, some sacrifices are made in terms of realism. To produce a more accurate tanker schedule, it would be advisable to make some of the following changes. Currently wind effects are not modeled. Adding wind effects would have dramatic impacts on the distance that aircraft can travel on a tankload of fuel, as well as effect the airspeed that an aircraft can travel. Including wind would alter the placement of refueling points as well as impact the time that these refueling points are serviced. The model currently has equations which include wind effects, as well as a worksheet for entering wind data; it is just a matter of determining the proper wind speed and directions to use. Additionally, this model does not account for aircraft having to change speed during air refueling, or account for the fact that refueling does not occur instantaneously. Normally, tankers decrease their cruise speed during refueling. Tankers then match the speed of fighter aircraft during the escort phase. A calculation would be required to incorporate these changes in speed to determine any modifications to the distance a tanker can travel

Due to current Air Force regulation, fighter aircraft require tanker escort over open bodies of water. However, in this model for simplicity, all fighter paths are assigned tanker escort, even over land. It would not be difficult to alter the model such that only paths over water get escorts. The model currently has code which could either call a database of land formations or some procedure which would determine whether or not the flight path between refueling points requires escort. Besides greater realism, this added improvement might decrease the number of tankers required for a given solution.

The model allows the user to define the flight path for a receiver group through the addition of in flight waypoints. Currently, only one waypoint can be entered for each receiver group. It is recommended that the model be upgraded so as to allow users to enter an unlimited number of waypoints. The added benefit of doing this is that doing so would allow for the model to account for restricted airspace. Additionally the use of waypoints is important, since most pilots prefer not to fly at extreme northern latitudes due to the lack of emergency landing locations.

Currently, the model only performs scheduling for receiver groups in a deployment role, that is for ferrying aircraft from one area of the world to another area. Tanker refueling can also be used in an employment role, increasing the range of strike aircraft. It would be highly recommended to add the ability to determine the refueling schedules in support of strike operations. The only changes needed to the model would be in the way refueling points are generated. The model currently has the ability to define air refueling points as well as allowing strike missions to be entered in the input screen.

Another addition of great importance would be the ability to visualize the generated schedules and routes. Gantt charts could be used to visualize the schedules. Some type of map function would be useful to see the actual receiver group routes as well as tanker routes. These two features would be useful in quickly determining if any errors had been made during the modeling process.

The code for this tool is written in Visual Basic for Applications (VBA) within Excel. In order to run a different deployment, the current set of receiver groups must be replaced with a new set. This includes the aircraft type, number of aircraft, base of origin

and destination, RLD and RDD. It would be nice to have the VBA program refer to another workbook containing the list of receiver groups. In this way, the tool would be independent of the input data and would only contain the aircraft performance and tanker location data.

Although most users are comfortable using Excel worksheets to manipulate data, modifying the code to Java would be an improvement. The Java code can be written to import the receiver group data, which the user inputs through an Excel interface. Java is platform independent and is object oriented. This object orientation could increase the manageability of the code and possibly decrease the computation time.

Another feature that could be added would be the ability to allow the user to determine what type of schedule they might wish. Currently the model finds the schedule that minimizes the flight distance of the tanker fleet. It would not be difficult to add the ability for a user to have the ability to minimize the schedules based not only on tanker flight distances but also by the total number of tankers required to support the operations. This might be desirable if you have severe tanker resource constraints.

A more robust scheduling routine would aid greatly in the performance of this tool. The least slack method is quite fast and easy to implement, however it does not necessarily get very good results. This is especially true when there are either limited tanker resources or when there are tight schedules between the inputted ready to load dates and required delivery dates. The other problem with the scheduling function is that it is not integrated within the ant colony metaheuristic. Therefore the number of tankers that the heuristic may use in finding the shortest tanker distance, may in fact be far greater than the number available. It is entirely possible that the ant colony may never

actually produce a feasible schedule that delivers all resources to their destination bases in time.

Finally, this model does not take into account tanker flight crew schedules. The number of flight crews would have a tremendous impact on tanker flight operations. In the model tanker routes may be scheduled which would not be possible if you consider tanker flight crew restrictions, such as maximum flight hours per day, maximum flight hours per month, or crew rest constraints. Scheduling flight crews is itself a very difficult process, however it may be possible to add some rules which would at least somewhat capture the idea of crew restrictions. Current AFIT research is being performed on the tanker crew scheduling problem. In the future, it may be possible to combine this model with that crew scheduling model to form a complete tanker scheduling tool.

Appendix A. TAP Tool VBA Coding

```
'Module:       Plan
'Author:       Capt RonJon Annaballi, USAF AFIT/ENS
'Last Updated:
'Function:     This module contains the primary code for the Quick Look Tanker Deployment
'              Tool.  'Sub Plan()' is run when the "Develop Master Plan" button located on the
'              Menu sheet is pushed

Public RParray(), RPdist(), PT(), Hold(), TnkSpeed(), TnkTime(), TnkParams() As Variant
Public RefuelPoints(), pheromone(), RPList(), TankerFuel(), AntTravel, RefuelInfo() As Variant
Public numAnts, RPnum, RPvisited, numTankerBases, numTankertypes, alpha, beta, numreceivergroups As Integer
Public antroutes(), bestroute(), tnkroutes(), besttnkroutes(), RPtime(), RGtiming(), route As Variant
Public MissionPlan(), Fuelrqmts(), tankBases(), routeRG(), RGarray, tnkAssign(), RouteInfo() As Variant
Public tankersched(), tnkstart(), tnkfinish(), NewAntRoute(), newtnkroute(), tnkcount() As Variant
Public numtanker As Variant


Sub Plan()


' Initialize some variables.

gettime = Time      'gettime stores current time (used to calculate running time of program)
rpcount = 0         'RPcount stores the number of refueling points needed for scenario
numAnts = 0
numIterations = 1
alpha = 1               'sets alpha rate for ACO heuristic
beta = 5                'sets beta rate for ACO heuristic
evaporation = 0.5       'sets evaporation rate for ACO heuristic



'SECTION I.  INPUT DATA

'i.    Assign user input from INPUT worksheet to data array 'ReceiverDdata'
'      This assignment does not put limits on the number of mission plans that can be entered.
'      For data processing the number of mission plans entered by the user
'      is counted and assigned to variable 'NumReceiverGroups'

Set tempfile = Sheets("INPUT").Range("A6").CurrentRegion
Receiverdata = tempfile.Offset(1, 0).Resize(tempfile.Rows.Count - 1, tempfile.Columns.Count)
numreceivergroups = tempfile.Rows.Count - 1

'ii.   Assign user input wind data from WINDS worksheet to data array 'winds'

'Set tempfile = Sheets("WINDS").Range("A7").CurrentRegion
'winds = tempfile.Offset(1, 0).Resize(tempfile.Rows.Count - 1, tempfile.Columns.Count)


'iii.  Assign ICAO data list in the AIRBASES worksheet to data array 'GlobalBaseList'. The number
'      of bases currently listed is counted and assigned to the variable 'NumGlobalBases'

Set tempfile = Sheets("AIRBASES").Range("A3").CurrentRegion
globalbaselist = tempfile.Offset(1, 0).Resize(tempfile.Rows.Count - 1, tempfile.Columns.Count)
NumGlobalBases = tempfile.Rows.Count - 1

'iv.   Assign RECEIVER DATA information to data array 'receivers' The number
'     of aircraft currently listed is counted and assigned to the variable 'numreceivertypes'

Set tempfile = Sheets("RECEIVER DATA").Range("A8").CurrentRegion
receivers = tempfile.Offset(2, 0).Resize(tempfile.Rows.Count - 2, tempfile.Columns.Count)
Numreceivertypes = tempfile.Rows.Count - 2

'v.  Assign TANKER DATA information to data array 'tankers' The number of
'     aircraft currently listed is counted and assigned to the variable 'numtankertypes'
```

```
Set tempfile = Sheets("TANKER DATA").Range("A4").CurrentRegion
tankers = tempfile.Offset(1, 0).Resize(tempfile.Rows.Count - 1, tempfile.Columns.Count)
numTankertypes = tempfile.Rows.Count - 1

ReDim TnkSpeed(1 To numTankertypes)

For i = 1 To numTankertypes
   TnkSpeed(i) = tankers(i, 3)
Next i

'vi.   Assign tanker base input from TankerBases worksheet to data array 'TankerBases. The number
'      of bases currently listed is counted and assigned to the variable 'NumTankerBases'

Set tempfile = Sheets("TankerBases").Range("A3").CurrentRegion
tankerBases = tempfile.Offset(1, 0).Resize(tempfile.Rows.Count - 1, tempfile.Columns.Count)
numTankerBases = tempfile.Rows.Count - 1

' Finds the index (row number) for each tanker base and assigns to RowIndex. Then assigns
' appropriate number of tankers to respective base in GlobalBaseList.  Finally an
' array is built for the tanker bases being used

ReDim TnkParams(1 To numTankerBases, 4)
ReDim tankBases(1 To numTankerBases, 6)
For n = 1 To numTankerBases
   tnkindex = Find(tankerBases(n, 3), tankers, numTankertypes)
   RowIndex = Find(tankerBases(n, 1), globalbaselist, NumGlobalBases)
   globalbaselist(RowIndex, 6) = tankerBases(n, 2)

   tankBases(n, 1) = globalbaselist(RowIndex, 1)    ' Base ID
   tankBases(n, 2) = globalbaselist(RowIndex, 2)    ' Base Name
   tankBases(n, 3) = globalbaselist(RowIndex, 4)    ' Base Latitude
   tankBases(n, 4) = globalbaselist(RowIndex, 5)    ' Base Longitude
   tankBases(n, 5) = globalbaselist(RowIndex, 6)    ' # of tankers at Base
   tankBases(n, 6) = tankerBases(n, 3)              ' type of tankers at Base

   TnkParams(n, 1) = tankers(tnkindex, 4) / tankers(tnkindex, 3)   'fuel flow conversion factor (=fuel flow/air speed)
   TnkParams(n, 2) = tankers(tnkindex, 7)                          'climb fuel
   TnkParams(n, 3) = tankers(tnkindex, 5) - tankers(tnkindex, 6)   'maximum fuel
   TnkParams(n, 4) = tankers(tnkindex, 3)                          'air speed

Next n

'vii. Checks Input Data

For i = 1 To numreceivergroups

   RowIndex = Find(Receiverdata(i, 2), receivers, Numreceivertypes)

   ' This section corrects the input data.  It compares the number of assets in the receiver group
   ' against the defined receiver group size in RECEIVER DATA worksheet.  If the input
   ' inputted size is larger, this section breaks up the receiver group into a number of smaller groups

   n = 1
   While Sheets("INPUT").Cells(i + 6, 3) > receivers(RowIndex, 2)

      Sheets("Input").Select
      Range(Cells(i + 6, 1), Cells(i + 6, 18)).Copy
      Sheets("INPUT").Paste Destination:=Sheets("INPUT").Range(Cells(numreceivergroups + 7, 1), Cells(numreceivergroups + 7,
18))
      Sheets("INPUT").Cells(i + 6, 3) = Sheets("INPUT").Cells(i + 6, 3) - receivers(RowIndex, 2)
      Sheets("INPUT").Cells(numreceivergroups + 7, 3) = receivers(RowIndex, 2)
      Sheets("INPUT").Cells(numreceivergroups + 7, 1) = Sheets("INPUT").Cells(numreceivergroups + 7, 1) & n
      n = n + 1
      numreceivergroups = numreceivergroups + 1
   Wend

Next i
```

```
' Reads in updated Input File

Set tempfile = Sheets("INPUT").Range("A6").CurrentRegion
Receiverdata = tempfile.Offset(1, 0).Resize(tempfile.Rows.Count - 1, tempfile.Columns.Count)


'SECTION II.  MISSION PLAN WORKSHEET

ReDim MissionPlan(1 To numreceivergroups, 15)
ReDim Fuelrqmts(1 To numreceivergroups, 10)
ReDim RefuelInfo(1 To numreceivergroups, 0 To 100)
ReDim RGtiming(1 To numreceivergroups, 4)
For i = 1 To numreceivergroups

   WayPointDist = 0        'Clears out past values
   firstrpdist = 0

   RGtiming(i, 1) = 24 * (Receiverdata(i, 12) - 1) 'converts ALD to hours
   RGtiming(i, 2) = 24 * (Receiverdata(i, 13) - 1) 'converts RDD to hours

'i.  Transfer the Sortie ID from user input to mission plan data array 'MissionPlan'

   MissionPlan(i, 1) = Receiverdata(i, 1)

'ii. Identify the latitude and longitude of user input origin and destination airbases
'    and assign to mission plan data array 'MissionPlan'

   RowIndex = Find(Receiverdata(i, 4), globalbaselist, NumGlobalBases) 'Origin Coordinates

   MissionPlan(i, 2) = globalbaselist(RowIndex, 4)
   MissionPlan(i, 3) = globalbaselist(RowIndex, 5)

   RowIndex = Find(Receiverdata(i, 5), globalbaselist, NumGlobalBases) 'Destination Coordinates

   MissionPlan(i, 4) = globalbaselist(RowIndex, 4)
   MissionPlan(i, 5) = globalbaselist(RowIndex, 5)

   If Receiverdata(i, 14) <> 0 Then              'Waypoint Coordinates (if any)
      MissionPlan(i, 14) = Receiverdata(i, 14)
      MissionPlan(i, 15) = Receiverdata(i, 15)
   End If

'iii. Compute the flight window (in days.  Subtracts "ready for departure date" from "required deliver date"
   MissionPlan(i, 6) = Receiverdata(i, 13) - Receiverdata(i, 12)

'iv.  Calculate the flight distance for the receiver groups

   If Receiverdata(i, 6) > 0 Then          'use for Receiver Groups that fly to an orbit point
      MissionPlan(i, 7) = Receiverdata(i, 6)

   ElseIf Receiverdata(i, 14) = "" Then      'use for Receiver Groups that fly directly to destination base
      MissionPlan(i, 7) = GreatCircleDistance(MissionPlan(i, 2), MissionPlan(i, 3), MissionPlan(i, 4), MissionPlan(i, 5))

   Else    'Use for Receiver Groups that fly to a waypoint along their path.  Calculates distance from origin
         'to waypoint.  Then does another calculation from waypoint to destination base and adds the two figures.

      WayPointDist = GreatCircleDistance(MissionPlan(i, 2), MissionPlan(i, 3), MissionPlan(i, 14), MissionPlan(i, 15))
      MissionPlan(i, 7) = WayPointDist + GreatCircleDistance(MissionPlan(i, 14), MissionPlan(i, 15), MissionPlan(i, 4),
MissionPlan(i, 5))

   End If

'v.   Referencing the RECEIVER DATA worksheet, the escort requirement and number of
'     receivers per tanker values are determined and assigned to the MissionPlan data array

   RowIndex = Find(Receiverdata(i, 2), receivers, Numreceivertypes)
   MissionPlan(i, 8) = receivers(RowIndex, 9)              'Escort Requirement
```

```
    MissionPlan(i, 9) = receivers(RowIndex, 2)              'Receiver Group max size


'vi.  Using the latitudes and longitudes of the origin and destination bases, the mean true
'     course of the receiver is calculated.  The mean true course, air speed, wind direction,
'     and wind velocity are then used to calculate the receiver ground speed.
'     Note:  Currently wind is not accounted for in the model, therefore this is not performed.

    ' TC = TrueCourse(MissionPlan(i, 7), MissionPlan(i, 2), MissionPlan(i, 3), MissionPlan(i, 4), MissionPlan(i, 5))
    ' grdspd = GroundSpeed(rdata(i, 7), TC, winds(i, 1), winds(i, 2))

    'Calculate maximum receiver flight duration (= great circle distance / ground speed).
    MissionPlan(i, 10) = MissionPlan(i, 7) / Receiverdata(i, 7)      'Max Duration Time
    MissionPlan(i, 11) = Receiverdata(i, 7)                          'True ground speed (= True Air speed if no wind)


    'SECTION III.  OFFLOAD CALCULATIONS

    'Fuelrqmts contains information to calculate fuel waypoints

    Fuelrqmts(i, 1) = Receiverdata(i, 1)          ' Sortie ID
    Fuelrqmts(i, 2) = Receiverdata(i, 11)         ' Total Fuel Capacity
    Fuelrqmts(i, 3) = Receiverdata(i, 7)          ' True air speed
    Fuelrqmts(i, 4) = MissionPlan(i, 11)          ' Ground Speed
    Fuelrqmts(i, 5) = receivers(RowIndex, 7)      ' Climb Fuel
    Fuelrqmts(i, 6) = receivers(RowIndex, 4)      ' Fuel Flow
    Fuelrqmts(i, 7) = receivers(RowIndex, 6)      ' Fuel Reserve

    FuelFlow1 = receivers(RowIndex, 12)       'Reads in fuel flow coefficients from RECEIVER DATA
    FuelFlow2 = receivers(RowIndex, 13)       'Used to find the fuel burn rate
    FuelFlow3 = receivers(RowIndex, 14)
    FuelFlow4 = receivers(RowIndex, 15)

    'Calculates refueling requirement.  Uses a third order polynomial function to calulate total fuel needed
    'Subtracts out the amount of starting fuel able to be used.  If the final number is negative then no
    'refueling is needed.  Fuelneeded is the amount of refueling needed for the receiver group.

    totalfuelneeded = recflburn(MissionPlan(i, 7), Fuelrqmts(i, 4), Fuelrqmts(i, 2), Fuelrqmts(i, 7), Receiverdata(i, 9), Receiverdata(i,
10), Fuelrqmts(i, 5), FuelFlow1, FuelFlow2, FuelFlow3, FuelFlow4)
    fuelneeded = Receiverdata(i, 3) * (totalfuelneeded - Fuelrqmts(i, 2) + Fuelrqmts(i, 7))
    If fuelneeded < 0 Then
       Fuelrqmts(i, 8) = 0
    Else
       Fuelrqmts(i, 8) = fuelneeded
    End If

    Fuelrqmts(i, 9) = totalfuelneeded       'Total Fuel Needed to Complete Trip

    ' Calculates maximum distance between refueling legs.
    '   Formula: (Fuel Capacity - Fuel Reserve)*Ground Speed/Fuel Flow

    Fuelrqmts(i, 10) = (Fuelrqmts(i, 2) - Fuelrqmts(i, 7)) * Fuelrqmts(i, 4) / Fuelrqmts(i, 6)

'SECTION IV.  Calculations for the refueling points.

    RefuelInfo(i, 0) = MissionPlan(i, 8)      ' Sortie ID
    RefuelInfo(i, 1) = Receiverdata(i, 1)     ' Sortie ID
    RefuelInfo(i, 2) = MissionPlan(i, 2)      ' Origin Latitude
    RefuelInfo(i, 3) = MissionPlan(i, 3)      ' Origin Longitude
    RefuelInfo(i, 4) = MissionPlan(i, 4)      ' Destination Latitude
    RefuelInfo(i, 5) = MissionPlan(i, 5)      ' Destination Longitude
    RefuelInfo(i, 6) = Fuelrqmts(i, 10)       ' Distance between RPs

    ' Calculates amount of fuel needed.  This assumes aircraft travel 100 miles while climbing.
    ' Fuelreq = climb fuel + amount of fuel needed to travel the total distance (minus 100 miles))

    fuelreq = Fuelrqmts(i, 5) + (MissionPlan(i, 7) - 100) * Fuelrqmts(i, 6) / Fuelrqmts(i, 4)
```

```
' Checks to see if refueling is needed. {yes if: fuelrequired > (fuel capacity - fuel reserve)}
' If so then determine first refueling point and the total amount of fuel needed at that point.

If fuelreq <= Fuelrqmts(i, 2) - Fuelrqmts(i, 7) Then GoTo NoRefuel  'If no refueling needed skip to next receiver group

If RefuelInfo(i, 6) > MissionPlan(i, 7) Then   'If max leg is greater than total flight distance

   firstrpdist = 100      'Set refueling point to TopOff Point (100 miles)
   RefuelInfo(i, 8) = Receiverdata(i, 3) * Fuelrqmts(i, 5)   'Refueling = # receivers * climb fuel

ElseIf Receiverdata(i, 16) > 0 Then   ' if input sheet has 1st refueling point overide

   firstrpdist = Receiverdata(i, 16)  ' set 1st refueling point to user input.
   ' Fuel = # receivers *(climb fuel + (distance to refuel point - climb distance)*fuel flow/speed))
   RefuelInfo(i, 8) = Receiverdata(i, 3) * (Fuelrqmts(i, 5) + (Receiverdata(i, 16) - 100) * Fuelrqmts(i, 6) / Fuelrqmts(i, 4))

Else    '1st ref dist = dist between RPs-(climb fuel*ground speed/fuel flow)
   firstrpdist = RefuelInfo(i, 6) - (Fuelrqmts(i, 5) * Fuelrqmts(i, 4) / Fuelrqmts(i, 6))
   ' Fuel = # receivers *(fuel capacity - fuel reserve)
   RefuelInfo(i, 8) = Receiverdata(i, 3) * (Fuelrqmts(i, 2) - Fuelrqmts(i, 7))

End If
rpcount = rpcount + 1

' Calculates the number of additional legs needed
RefuelInfo(i, 7) = Ceiling((MissionPlan(i, 7) - firstrpdist) / RefuelInfo(i, 6) - 1)

' If only 1 refueling point is needed, then correct offload amount to equal fuel needed to get to final base
' fuel = # receivers *(total distance - distance to 1st RP)* fuel flow/speed
If RefuelInfo(i, 7) = 0 Then

   RefuelInfo(i, 8) = Receiverdata(i, 3) * (MissionPlan(i, 7) - firstrpdist) * Fuelrqmts(i, 6) / Fuelrqmts(i, 4)

End If

' Calculates the coordinates and azimuth to first refueling point.

If firstrpdist > 0 Then    ' If the distance to first RP is greater than 0.

   If WayPointDist = 0 Then      ' For no defined waypoints

      azym = getAz(RefuelInfo(i, 2), RefuelInfo(i, 3), RefuelInfo(i, 4), RefuelInfo(i, 5))
      RefuelInfo(i, 9) = getLat(RefuelInfo(i, 2), firstrpdist, azym)
      RefuelInfo(i, 10) = getLong(RefuelInfo(i, 3), firstrpdist, azym, RefuelInfo(i, 9))

   ElseIf firstrpdist < WayPointDist Then  'If waypoint is after first refuel point

      azym = getAz(RefuelInfo(i, 2), RefuelInfo(i, 3), Receiverdata(i, 14), Receiverdata(i, 15))
      RefuelInfo(i, 9) = getLat(RefuelInfo(i, 2), firstrpdist, azym)
      RefuelInfo(i, 10) = getLong(RefuelInfo(i, 3), firstrpdist, azym, RefuelInfo(i, 9))

   Else    ' If 1st RP is after the waypoint

      azym = getAz(Receiverdata(i, 14), Receiverdata(i, 15), RefuelInfo(i, 4), RefuelInfo(i, 5))
      dist = firstrpdist - WayPointDist
      RefuelInfo(i, 9) = getLat(Receiverdata(i, 14), dist, azym)
      RefuelInfo(i, 10) = getLong(Receiverdata(i, 15), dist, azym, RefuelInfo(i, 9))

   End If
End If

'Determine the minimum number of tankers that can service this refuel point

tankersneeded = 100    'Initialize number of tankers needed to be very large
For t = 1 To numTankerBases
' Calculates distance between RP and tankerbase, finds the tanker type, and then calculates the roundtrip fuel
' needed to reach refueling point and then return to base (Stores the fuel needed as FuelTravel)
```

```
    dist = GreatCircleDistance(RefuelInfo(i, 9), RefuelInfo(i, 10), tankBases(t, 3), tankBases(t, 4))
    RowIndex = Find(tankBases(t, 6), tankers, numTankertypes)
    FuelTravel = tankers(RowIndex, 7) + (2 * dist - 100) * tankers(RowIndex, 4) / tankers(RowIndex, 3)
    'Calculate the offload available at the RP
    maxtankeroffload = tankers(RowIndex, 5) - tankers(RowIndex, 6) - FuelTravel
    'If tanker can refuel, calculate the number needed to meet this RP requirement
    If maxtankeroffload > 0 Then
       required = Ceiling(RefuelInfo(i, 8) / maxtankeroffload)
       If required < tankersneeded Then tankersneeded = required
    End If

 Next t



 'Breaks up large refueling points into smaller pieces based on the minimum number of tankers that can reach
 'that refueling point and return to its tanker base

 If tankersneeded > 1 Then
    offload = RefuelInfo(i, 8) / tankersneeded
    RefuelInfo(i, 8) = offload
    For k = 1 To (tankersneeded - 1)     'Since we already have first part of RP stored, only need to add the remaining sections
       RefuelInfo(i, 8 + 3 * k) = offload
       RefuelInfo(i, 9 + 3 * k) = RefuelInfo(i, 9)
       RefuelInfo(i, 10 + 3 * k) = RefuelInfo(i, 10)
       rpcount = rpcount + 1   'Increments RP counter
    Next k
    ArrayOffset = tankersneeded - 1
 'if refueling point offload is greater than 70 then split in half even if one tanker can reach it
 ElseIf RefuelInfo(i, 8) > 70 Then
    RefuelInfo(i, 8) = RefuelInfo(i, 8) / 2
    RefuelInfo(i, 8 + 3) = RefuelInfo(i, 8)
    RefuelInfo(i, 9 + 3) = RefuelInfo(i, 9)
    RefuelInfo(i, 10 + 3) = RefuelInfo(i, 10)
    rpcount = rpcount + 1   'Increments RP Counter
    ArrayOffset = 1
 Else    'No adjustment of refueling point is needed
    ArrayOffset = 0
 End If

 ' Checks escort requirement.  If needed construct a new RP with same coordinates as current RP but with zero offload

 If MissionPlan(i, 8) = "Y" Or MissionPlan(i, 8) = "y" Then

    ' Subroutine call to check if Refueling Point is over water would be located here

    ArrayOffset = ArrayOffset + 1
    RefuelInfo(i, 8 + 3 * ArrayOffset) = 0                 'Sets offload
    RefuelInfo(i, 9 + 3 * ArrayOffset) = RefuelInfo(i, 9)   'Sets latitude
    RefuelInfo(i, 10 + 3 * ArrayOffset) = RefuelInfo(i, 10) 'Sets longitude
    rpcount = rpcount + 1   'Increments RP Counter
 End If


' Calculate Remaining Refueling points

For j = 1 To RefuelInfo(i, 7)
       AO = ArrayOffset + 1
       ' find the coordinates for the next RP
       If WayPointDist = 0 Then    'For no defined waypoint
          azym = getAz(RefuelInfo(i, 9 + 3 * (AO - 1)), RefuelInfo(i, 10 + 3 * (AO - 1)), RefuelInfo(i, 4), RefuelInfo(i, 5))
          RefuelInfo(i, 9 + 3 * AO) = getLat(RefuelInfo(i, 9 + 3 * (AO - 1)), RefuelInfo(i, 6), azym)
          RefuelInfo(i, 10 + 3 * AO) = getLong(RefuelInfo(i, 10 + 3 * (AO - 1)), RefuelInfo(i, 6), azym, RefuelInfo(i, 9 + 3 * AO))

       ElseIf firstrpdist + j * RefuelInfo(i, 6) < WayPointDist Then 'If waypoint is after this refuel point
          azym = getAz(RefuelInfo(i, 9 + 3 * (AO - 1)), RefuelInfo(i, 10 + 3 * (AO - 1)), Receiverdata(i, 14), Receiverdata(i, 15))
          RefuelInfo(i, 9 + 3 * AO) = getLat(RefuelInfo(i, 9 + 3 * (AO - 1)), RefuelInfo(i, 6), azym)
          RefuelInfo(i, 10 + 3 * AO) = getLong(RefuelInfo(i, 10 + 3 * (AO - 1)), RefuelInfo(i, 6), azym, RefuelInfo(i, 9 + 3 * AO))
```

```
        Else   ' If RP is after the waypoint
           azym = getAz(Receiverdata(i, 14), Receiverdata(i, 15), RefuelInfo(i, 4), RefuelInfo(i, 5))
           dist = (firstrpdist + j * RefuelInfo(i, 6)) - WayPointDist
           RefuelInfo(i, 9 + 3 * AO) = getLat(Receiverdata(i, 14), dist, azym)
           RefuelInfo(i, 10 + 3 * AO) = getLong(Receiverdata(i, 15), dist, azym, RefuelInfo(i, 9 + 3 * AO))

        End If
        If j < RefuelInfo(i, 7) Then   'If not the last refuel point; fuel needed = # receivers *(fuel capacity - fuel reserve)
           RefuelInfo(i, 8 + 3 * AO) = Receiverdata(i, 3) * (Fuelrqmts(i, 2) - Fuelrqmts(i, 7))

        Else ' for last refuel point, only need enough fuel to get to final base
            ' fuel = # receivers *(total distance - (distance to 1st RP + distance to final RP))* fuel flow/speed

           RefuelInfo(i, 8 + 3 * AO) = Receiverdata(i, 3) * (MissionPlan(i, 7) - (firstrpdist + j * RefuelInfo(i, 6))) * Fuelrqmts(i, 6) /
Fuelrqmts(i, 4)

        End If

        rpcount = rpcount + 1   'Increments the counter for number of refueling points needed

        'Determine the minimum number of tankers that can service this refuel point

        tankersneeded = 100    'Initialize number of tankers needed to be very large
        For t = 1 To numTankerBases
        ' Calculates distance between RP and tankerbase, finds the tanker type, and then calculates the roundtrip fuel
        ' needed to reach refueling point and then return to base (Stores the fuel needed as FuelTravel)
           dist = GreatCircleDistance(RefuelInfo(i, 9 + 3 * AO), RefuelInfo(i, 10 + 3 * AO), tankBases(t, 3), tankBases(t, 4))
           RowIndex = Find(tankBases(t, 6), tankers, numTankertypes)
           FuelTravel = tankers(RowIndex, 7) + (2 * dist - 100) * tankers(RowIndex, 4) / tankers(RowIndex, 3)
           'Calculate the offload available at the RP
           maxtankeroffload = tankers(RowIndex, 5) - tankers(RowIndex, 6) - FuelTravel
           'If tanker can refuel, calculate the number needed to meet this RP requirement
           If maxtankeroffload > 0 Then
              required = Ceiling(RefuelInfo(i, 8 + 3 * AO) / maxtankeroffload)
              If required < tankersneeded Then tankersneeded = required
           End If

        Next t

        'Breaks up large refueling points into smaller pieces based on the minimum number of tankers that can reach
        'that refueling point and return to its tanker base

        If tankersneeded > 1 Then
           offload = RefuelInfo(i, 8 + 3 * AO) / tankersneeded
           RefuelInfo(i, 8 + 3 * AO) = offload
           For k = 1 To (tankersneeded - 1)
              RefuelInfo(i, 8 + 3 * (AO + k)) = offload
              RefuelInfo(i, 9 + 3 * (AO + k)) = RefuelInfo(i, 9 + 3 * AO)
              RefuelInfo(i, 10 + 3 * (AO + k)) = RefuelInfo(i, 10 + 3 * AO)
              rpcount = rpcount + 1   'Increments RP counter
           Next k
           ArrayOffset = AO + tankersneeded - 1
        'if refueling point offload is greater than 70 then split in half even if one tanker can reach it
        ElseIf RefuelInfo(i, 8 + 3 * AO) > 70 Then
           RefuelInfo(i, 8 + 3 * AO) = RefuelInfo(i, 8 + 3 * AO) / 2
           RefuelInfo(i, 8 + 3 * (AO + 1)) = RefuelInfo(i, 8 + 3 * AO)
           RefuelInfo(i, 9 + 3 * (AO + 1)) = RefuelInfo(i, 9 + 3 * AO)
           RefuelInfo(i, 10 + 3 * (AO + 1)) = RefuelInfo(i, 10 + 3 * AO)
           rpcount = rpcount + 1   'Increments RP Counter
           ArrayOffset = AO + 1
        Else    'No adjustment of refueling point is needed
           ArrayOffset = AO
        End If

        ' Checks escort requirement.  If needed construct a new RP with same coordinates as current RP but with zero offload

        If MissionPlan(i, 8) = "Y" Or MissionPlan(i, 8) = "y" Then
```

```
        ' Subroutine call to check if Refueling Point is over water would be located here

            ArrayOffset = ArrayOffset + 1
            RefuelInfo(i, 8 + 3 * ArrayOffset) = 0                      'Sets offload
            RefuelInfo(i, 9 + 3 * ArrayOffset) = RefuelInfo(i, 9 + 3 * AO)   'Sets latitude
            RefuelInfo(i, 10 + 3 * ArrayOffset) = RefuelInfo(i, 10 + 3 * AO) 'Sets longitude
            rpcount = rpcount + 1   'Increments RP Counter
         End If

   Next j
NoRefuel:   'If no refueling is needed for a receiver group, skip to here

Next i

'-------------------------------------------------------------------------------------------------------------

' Creates array which stores Refueling Point lat/long and offload required

ReDim RefuelPoints(1 To rpcount, 12)

k = 0
For i = 1 To numreceivergroups
   j = 0
   While RefuelInfo(i, 8 + 3 * j) <> ""
      k = k + 1
      If RefuelInfo(i, 0) = "y" Or RefuelInfo(i, 0) = "Y" Then
         RefuelPoints(k, 9) = 1
      Else
         RefuelPoints(k, 9) = 0
      End If             'escort requirement
      RefuelPoints(k, 1) = RefuelInfo(i, 1)             'Receiver Group Name
      RefuelPoints(k, 2) = RefuelInfo(i, 9 + 3 * j)      'latitude of RP
      RefuelPoints(k, 3) = RefuelInfo(i, 10 + 3 * j)     'longitude of RP
      RefuelPoints(k, 4) = RefuelInfo(i, 8 + 3 * j)      'offload at RP
      j = j + 1

      'determines the time windows for the RPs.
      '   RefuelPoints(k, 5) represents the earliest possible start (arrival at refueling point)
      '   RefuelPoints(k, 5) represents the latest possible start (arrival at refueling point)

      dist = GreatCircleDistance(MissionPlan(i, 2), MissionPlan(i, 3), RefuelPoints(k, 2), RefuelPoints(k, 3))

      RefuelPoints(k, 5) = (Receiverdata(i, 12) - 1) * 24 + (dist / MissionPlan(i, 11))
      RefuelPoints(k, 6) = (Receiverdata(i, 13) - 1) * 24 - ((MissionPlan(i, 7) - dist) / MissionPlan(i, 11))
   Wend
Next i

'Defines the RP: Receiver Group name and the actual RP number for that Receiver Group
i = 1
RefuelPoints(1, 7) = RefuelPoints(1, 1) & i
RefuelPoints(1, 10) = 1
RGnum = 1
RefuelPoints(1, 11) = RGnum

For k = 2 To rpcount
   If RefuelPoints(k, 2) = RefuelPoints(k - 1, 2) And RefuelPoints(k, 3) = RefuelPoints(k - 1, 3) And RefuelPoints(k, 1) =
RefuelPoints(k - 1, 1) Then
      RefuelPoints(k, 7) = RefuelPoints(k, 1) & i
      RefuelPoints(k, 10) = i
   Else
      i = i + 1
      RefuelPoints(k, 7) = RefuelPoints(k, 1) & i
      RefuelPoints(k, 10) = i
   End If
   If RefuelPoints(k, 1) <> RefuelPoints(k - 1, 1) Then
      i = 1
      RGnum = RGnum + 1
```

```
        RefuelPoints(k, 7) = RefuelPoints(k, 1) & i
        RefuelPoints(k, 10) = i
    End If
    RefuelPoints(k, 11) = RGnum
Next k

' Finds the escort RP associated with each offload RP
For k = rpcount To 2 Step -1
    If RefuelPoints(k, 4) = 0 Then
        escort = k
        RefuelPoints(k, 8) = escort
    End If
    If RefuelPoints(k, 7) = RefuelPoints(k - 1, 7) And (RefuelPoints(k - 1, 9) = 1 Or RefuelPoints(k - 1, 9) = 1) Then
        RefuelPoints(k - 1, 8) = escort
    End If
Next k


' Creates arrays which store the distances between refueling points and between tankerbases and refueling points
' Then creates array "TankerFuel" which stores the amount of fuel needed to get from tankerbase to refuel point

ReDim RPdist(1 To rpcount, 1 To rpcount + numTankerBases)
ReDim TankerFuel(1 To numTankerBases, 1 To rpcount)

For i = 1 To rpcount

    For j = i + 1 To rpcount
        RPdist(i, j) = GreatCircleDistance(RefuelPoints(i, 2), RefuelPoints(i, 3), RefuelPoints(j, 2), RefuelPoints(j, 3))
        RPdist(j, i) = RPdist(i, j)
    Next j

    For k = 1 To numTankerBases    'Finds roundtrip fuel required from tankerbase k to refuel point i

        RPdist(i, rpcount + k) = GreatCircleDistance(RefuelPoints(i, 2), RefuelPoints(i, 3), tankBases(k, 3), tankBases(k, 4))
        tnkrrndtrpdist = 2 * RPdist(i, rpcount + k)
        RowIndex = Find(tankBases(k, 6), tankers, numTankertypes)   'Determines the type of tanker available
        TankerFuel(k, i) = tankers(RowIndex, 7) + (tnkrrndtrpdist - 100) * tankers(RowIndex, 4) / tankers(RowIndex, 3)

    Next k

Next i
'-------------------------------------------------------------------------------------------------
'For each tanker base, apply the following ACO Heuristic
ReDim pheromone(1 To rpcount, 1 To rpcount + numTankerBases)
ReDim RParray(1 To rpcount)


For i = 1 To numTankerBases

    RowIndex = Find(tankBases(i, 6), tankers, numTankertypes)   'Determines the tanker type at base
    For k = 1 To rpcount
        'Checks to see if tanker can make roundtrip journey and meet offload requirement
            'Roundtrip fuel < max capacity - fuel reserve - required offload
        If TankerFuel(i, k) <= (tankers(RowIndex, 5) - tankers(RowIndex, 6) - RefuelPoints(k, 4)) Then

            pheromone(k, rpcount + i) = 10000 / (2 * RPdist(k, rpcount + i)) 'sets initial pheromone trail to roundtrip distance

        End If

    Next k
Next i

For x = 1 To numIterations

  ReDim Hold(1 To rpcount, 1 To rpcount + numTankerBases)     'Clears out holding container

  For i = 1 To numTankerBases   'Determine which RPs are in range of tanker base and store in RParray
```

```
   RowIndex = Find(tankBases(i, 6), tankers, numTankertypes)   'Determines the tanker type at base
   RPnum = 0
   For k = 1 To rpcount
      'Checks to see if tanker can make roundtrip journey and meet offload requirement
         'Roundtrip fuel < max capacity - fuel reserve - required offload
      If TankerFuel(i, k) <= (tankers(RowIndex, 5) - tankers(RowIndex, 6) - RefuelPoints(k, 4)) Then

         RPnum = RPnum + 1
         RParray(k) = 1     'Specifies that the refueling point is within range

      Else
         RParray(k) = 0      'Specifies that the refueling point is out of range

      End If

   Next k

   'Builds ant routes (one at a time) starting with each RP in range of tanker

   For k = 1 To rpcount

      ReDim PT(1 To rpcount, 1 To rpcount + numTankerBases)   'Clears out PT array

      RPList = RParray
      If RParray(k) = 1 Then

         'Calculate fuel needed to get from tanker base to RP
         fuelused = tankers(RowIndex, 7) + (RPdist(k, rpcount + i) - 100) * TnkParams(i, 1) + RefuelPoints(k, 4)
         Call AntColony(i, k, fuelused, rpcount, 0) 'transfers control to the AntColony subroutine with current tankerbase and current
RP to start with

         'adjust AntTravel base on number of RPs visited
         AntTravel = (AntTravel + 1000 * (rpcount - RPvisited)) / 100000

         For n = 1 To rpcount    'updates pheromone trail for current colony
            For m = 1 To rpcount + numTankerBases
               Hold(n, m) = Hold(n, m) + PT(n, m) / AntTravel
            Next m
         Next n

      End If
   Next k

  Next i

   'update pheromone trail based on last iteration
   For n = 1 To rpcount    'updates pheromone trail for current colony
      For m = 1 To rpcount + numTankerBases
         pheromone(n, m) = evaporation * pheromone(n, m) + Hold(n, m)
      Next m
   Next n

Next x

'-------------------------------------------------------------------------------
'Calculates time to travel from each tankerbase to each refuel point
'Stores the time in TnkTime()

ReDim TnkTime(1 To numTankerBases, 1 To rpcount)

For i = 1 To numTankerBases
   RowIndex = Find(tankBases(i, 6), tankers, numTankertypes)
   For j = 1 To rpcount
      TnkTime(i, j) = RPdist(j, rpcount + i) / TnkSpeed(RowIndex)
   Next j
Next i

'-----------------------------------------------------------------------------------------------
```

```
'This section is used to generate routes based on all tanker bases

ReDim bestroute(1 To rpcount, 1 To rpcount) As Variant
ReDim besttnkroute(1 To rpcount) As Variant
ReDim besttnksched(4, 1 To rpcount) As Variant
ReDim RPtime(1 To rpcount) As Variant

bestSol = 1E+25     'initializes starting solution values
bestRP = -1
RPnum = rpcount     'makes all bases within range

For k = 1 To rpcount

   RParray(k) = 1      'Initializes the RParray to indicate all bases are within range
Next k

For x = 1 To numIterations

 ReDim Hold(1 To rpcount, 1 To rpcount + numTankerBases)     'Clears out holding container

   For k = 1 To rpcount

      ReDim PT(1 To rpcount, 1 To rpcount + numTankerBases)   'Clears out PT array

      RPList = RParray

      i = tankerFind(k, rpcount) 'Determines which tanker to use to service Refuel Point k
      fuelused = TnkParams(i, 2) + (RPdist(k, rpcount + i) - 100) * TnkParams(i, 1) + RefuelPoints(k, 4)

      'Call ACO heuristic to generate route
      Call AntColony(i, k, fuelused, rpcount, 1)

      'Calls scheduling algorithm

      Call schedule(rpcount)

      '---------------------------------------------------------------------
      'Evaluates route and schedule

      If AntTravel < bestSol And RPvisited >= bestRP And k = 1 Then
         bestRP = RPvisited
         bestSol = AntTravel
         bestroutes = route
         RPbest = k
         bestnumtankers = numtanker

         For n = 1 To rpcount
            For m = 1 To rpcount
               bestroute(n, m) = antroutes(n, m)
            Next m
            besttnkroute(n) = newtnkroute(n)
            RefuelPoints(n, 12) = RPtime(n)

            besttnksched(1, n) = tankersched(1, n)
            besttnksched(2, n) = tankersched(2, n)
            besttnksched(3, n) = tankersched(3, n)
            besttnksched(4, n) = tankersched(4, n)
         Next n
      End If

      '---------------------------------------------------------------------

      'adjust AntTravel base on number of RPs visited
      AntTravel = (AntTravel + 1000 * (rpcount - RPvisited)) / 100000

      For n = 1 To rpcount    'updates pheromone trail for current colony
         For m = 1 To rpcount + numTankerBases
            Hold(n, m) = Hold(n, m) + PT(n, m) / AntTravel
```

```
            Next m
        Next n

    Next k

    'update pheromone trail based on last iteration
    For n = 1 To rpcount    'updates pheromone trail for current colony
        For m = 1 To rpcount + numTankerBases
            pheromone(n, m) = evaporation * pheromone(n, m) + Hold(n, m)
        Next m
    Next n

Next x

'-------------------------------------------------------------------------------------------
'Outputs pertinent information to the worksheets

'Outputs information about the Refueling Points
Sheets("RP locations").Select
For n = 1 To rpcount

    Cells(n + 1, 1) = RefuelPoints(n, 1)    'RG Name
    Cells(n + 1, 2) = RefuelPoints(n, 10)   'RP #
    Cells(n + 1, 3) = RefuelPoints(n, 2)    'RP Lat
    Cells(n + 1, 4) = RefuelPoints(n, 3)    'RP long
    Cells(n + 1, 5) = RefuelPoints(n, 4)    'Offload
    Cells(n + 1, 6) = RefuelPoints(n, 5)    'Refuel Time
    Cells(n + 1, 8) = RGtiming(RefuelPoints(n, 11), 4)      'final arrival time at base

Next n

'Outputs information about the Tanker Routes
Sheets("Ants").Select
For i = 1 To bestroutes
    For j = 1 To rpcount
        Cells(i + 1, j + 3) = bestroute(i, j)
    Next j
    Cells(i + 1, 2) = RouteInfo(i, 7)
    Cells(i + 1, 3) = RouteInfo(i, 8)
    Cells(i + 1, 1) = tankBases(besttnkroute(i), 2) & " / " & RouteInfo(i, 9)
    Cells(i + 1, 11) = routeRG(i, 1)

Next i

Cells(bestroutes + 3, 4) = "Tankers Used: " & bestnumtankers
Cells(bestroutes + 4, 4) = "Total Tanker Distance: " & bestSol & " miles"


Sheets("Tankers").Select
For i = 1 To route

    Cells(1, i + 1) = besttnksched(1, i)
    Cells(2, i + 1) = besttnksched(2, i)
    Cells(3, i + 1) = besttnksched(3, i)
    Cells(4, i + 1) = besttnksched(4, i)

Next i

MsgBox (bestSol)
gettime = (Time - gettime)
Sheets("Menu").Cells(22, 2) = gettime
Sheets("Menu").Cells(24, 2) = numAnts
MsgBox ("I'm Done")

End Sub
Sub AntColony(tnkbase, startRP, startFuel, totalRP, ACOcall)

'This subroutine is the ACO heuristic used to determine the optimal routing for each tankerbase.
```

```
   ReDim antroutes(1 To totalRP, 1 To totalRP)
   ReDim tnkroutes(1 To totalRP) As Variant

   route = 1
   routestep = 1
   currentRP = startRP
   numRP = RPnum
   RPvisited = 0
   prevescort = -99
   antroutes(route, routestep) = currentRP     'Adds RP to route
   AntTravel = RPdist(currentRP, totalRP + tnkbase)
   PT(currentRP, totalRP + tnkbase) = 1

   If RefuelPoints(currentRP, 4) <> 0 Then     'Takes currentRP off active list if it is not an "escort" point
      RPList(currentRP) = 0
      numRP = numRP - 1
      RPvisited = RPvisited + 1
   End If

   If RefuelPoints(startRP, 9) = 1 And RefuelPoints(startRP, 10) > 1 Then
      'if currentRP is not associated with the first RP in an escort path
      temp = startRP - 1
      While RefuelPoints(startRP, 7) = RefuelPoints(temp, 7)
         temp = temp - 1
      Wend
      prevescort = RefuelPoints(temp, 8)          'finds the previous escort refueling point
   End If

   While numRP > 0

      bestSol = -999          'Initializes starting values
      nextrp = 0

      'Calculates probabilty function values for all possible moves from current RP
      'Then determines the best feasible move from current RP based on RGs involved, sequencing within RG
      'and type of aircraft involved

      For j = 1 To totalRP
         tnkroutes(route) = tnkbase
         If RPList(j) = 1 And j <> currentRP And Not (currentRP > RefuelPoints(j, 8) And RefuelPoints(currentRP, 1) =
RefuelPoints(j, 1)) Then
            'if the new RP is not used and it doesn't come before the currentRP in same RG then continue
            fuelused = startFuel + (RPdist(currentRP, j) + RPdist(j, totalRP + tnkbase)) * TnkParams(tnkbase, 1) + RefuelPoints(j, 4)
            If fuelused < TnkParams(tnkbase, 3) Then
               'if fuel needed for the route is less than max available for tanker then continue
               probfunction = pheromone(currentRP, j) ^ alpha + (10000 / (RPdist(currentRP, j) + 1)) ^ beta
               If probfunction > bestSol Then
                  'if prob function is better than any previous then continue
                  NxtEscort = RefuelPoints(j, 8)
                  crntescort = RefuelPoints(currentRP, 8)
                  followEscort = fighterescort(currentRP, totalRP)
                  If RefuelPoints(currentRP, 9) = 0 Or RefuelPoints(j, 9) = 0 Or RefuelPoints(currentRP, 1) <> RefuelPoints(j, 1) Or
RefuelPoints(currentRP, 7) = RefuelPoints(j, 7) Or followEscort = NxtEscort Then
                     'if either RP involved is a heavy or if RPs belong to same coordinates or new RP is next RP in sequence
                     If RefuelPoints(currentRP, 1) <> RefuelPoints(j, 1) Then 'Note: I deleted this from begining of line
(RefuelPoints(currentRP, 4) = 0 And)
                     Else
                        bestSol = probfunction
                        nextrp = j
                     End If

                  End If
               End If
            End If
         End If
      Next j
```

```
        'This ensures that the escort to the starting RP is made
        If nextrp = prevescort Then
           If RPList(RefuelPoints(startRP, 8)) = 0 Then
              RPList(RefuelPoints(startRP, 8)) = 1
              numRP = numRP + 1
              prevescort = ""
           End If
        End If

        If nextrp <> 0 Then         'There is a feasible point within range of current point

           If RefuelPoints(currentRP, 9) = 0 Or RefuelPoints(nextrp, 9) = 0 Or RefuelPoints(currentRP, 1) <> RefuelPoints(nextrp, 1)
Then
              'No Escort needed for this route
              routestep = routestep + 1
              antroutes(route, routestep) = nextrp
              AntTravel = AntTravel + RPdist(currentRP, nextrp)
              PT(currentRP, nextrp) = 1
              'If ACOcall = 1 Then MsgBox ("Route: " & Route & " Step: " & routestep & " RP: " & nextrp & " No escort")
              startFuel = startFuel + RPdist(currentRP, nextrp) * TnkParams(tnkbase, 1) + RefuelPoints(nextrp, 4)
              If RefuelPoints(nextrp, 4) <> 0 Then 'if nextRP is not an escort point then update the number of RPs remaining
                 numRP = numRP - 1
                 RPList(nextrp) = 0
                 RPvisited = RPvisited + 1
              End If
              currentRP = nextrp

           ElseIf RefuelPoints(currentRP, 7) = RefuelPoints(nextrp, 7) Then
              'No escort needed for this route.  Refuel points has same coordinates
              routestep = routestep + 1
              antroutes(route, routestep) = nextrp
              AntTravel = AntTravel + RPdist(currentRP, nextrp)
              PT(currentRP, nextrp) = 1
              'If ACOcall = 1 Then MsgBox ("Route: " & Route & " Step: " & routestep & " RP: " & nextrp & " same rp")
              startFuel = startFuel + RPdist(currentRP, nextrp) * TnkParams(tnkbase, 1) + RefuelPoints(nextrp, 4)
              If RefuelPoints(nextrp, 4) <> 0 Then    'if nextRP is not an escort point then update the number of RPs remaining
                 numRP = numRP - 1
                 RPList(nextrp) = 0
                 RPvisited = RPvisited + 1
              End If
              currentRP = nextrp

           Else    'Escort is needed

              crntescort = RefuelPoints(currentRP, 8)
              routestep = routestep + 1
              antroutes(route, routestep) = nextrp
              AntTravel = AntTravel + RPdist(currentRP, nextrp)
              PT(currentRP, nextrp) = 1
              startFuel = startFuel + RPdist(currentRP, nextrp) * TnkParams(tnkbase, 1) + RefuelPoints(nextrp, 4)
              'If ACOcall = 1 Then MsgBox ("Route: " & Route & " Step: " & routestep & " RP: " & nextrp & " escort needed")
              If RPList(currentRP) = 1 Then   'deletes currentRP from list if needed
                 numRP = numRP - 1
                 RPList(currentRP) = 0
                 RPvisited = RPvisited + 1
              End If

              If RefuelPoints(nextrp, 4) <> 0 Then 'if nextRP is not an escort, then take off list
                 numRP = numRP - 1
                 RPList(nextrp) = 0
                 RPvisited = RPvisited + 1
              End If
              currentRP = nextrp

           End If


        Else        'There are no valid points within range of current point
```

```
        If routestep <> 1 Then
            route = route + 1
            routestep = 1
            antroutes(route, routestep) = currentRP
            AntTravel = AntTravel + RPdist(currentRP, totalRP + tnkbase)    'adds in new route initial leg
            PT(currentRP, totalRP + tnkbase) = 1
            If ACOcall = 1 Then     'If considering all bases, then call function to find best start pt
                tnkbase = tankerFind(currentRP, totalRP)
            End If
            'If ACOcall = 1 Then MsgBox ("Route: " & Route & " Step: " & routestep & " RP: " & currentRP & " New route continue")
            startFuel = TnkParams(tnkbase, 2) + (RPdist(currentRP, totalRP + tnkbase) - 100) * TnkParams(tnkbase, 1) +
RefuelPoints(currentRP, 4)

        Else
            If RPList(currentRP) = 1 Then
                RPList(currentRP) = 0
                numRP = numRP - 1
                RPvisited = RPvisited + 1
            End If
            If RefuelPoints(currentRP, 4) <> 0 Then
                AntTravel = AntTravel + RPdist(currentRP, totalRP + tnkbase)    'adds in return leg
                route = route + 1
                RPvisited = RPvisited + 1
                PT(currentRP, totalRP + tnkbase) = 1
            Else
                antroutes(route, routestep) = ""    'clears out antroute for unneeded leg
                tnkroutes(route) = ""               'clears out tnkroutes for unneeded leg
                AntTravel = AntTravel - RPdist(currentRP, totalRP + tnkbase)    'subtracts unneeded leg
                PT(currentRP, totalRP + tnkbase) = 0
            End If

            If numRP <> 0 Then
                While RPList(currentRP) <> 1
                    currentRP = currentRP + 1
                    If currentRP > totalRP Then currentRP = 1
                Wend
                antroutes(route, routestep) = currentRP
                AntTravel = AntTravel + RPdist(currentRP, totalRP + tnkbase)    'adds in new route initial leg
                PT(currentRP, totalRP + tnkbase) = 1

                If ACOcall = 1 Then     'If considering all bases, then call function to find best start pt
                    tnkbase = tankerFind(currentRP, totalRP)
                End If
                'If ACOcall = 1 Then MsgBox ("Route: " & Route & " Step: " & routestep & " RP: " & currentRP & " new route new
point")
                If RefuelPoints(currentRP, 4) <> 0 Then
                    numRP = numRP - 1
                    RPList(currentRP) = 0
                    RPvisited = RPvisited + 1
                End If
                startFuel = TnkParams(tnkbase, 2) + (RPdist(currentRP, totalRP + tnkbase) - 100) * TnkParams(tnkbase, 1) +
RefuelPoints(currentRP, 4)

            End If
        End If
    End If

    'This ensures that the escort to the starting RP is made
    If currentRP = prevescort Then
        If RPList(RefuelPoints(startRP, 8)) = 0 Then
            RPList(RefuelPoints(startRP, 8)) = 1
            numRP = numRP + 1
            prevescort = ""
        End If
    End If

  Wend
  If antroutes(route, 1) = "" Then route = route - 1
```

```
    numAnts = numAnts + 1


End Sub

Sub schedule(totalRP)
'This subroutine is used to determine the schedules based on the routes generated by the ACO heuristic
'Uses "antroutes" and "tnkroutes" array generated by ACO heuristic
ReDim adjustment(1 To numreceivergroups) As Variant
ReDim routeRG(1 To totalRP, 3) As Variant
ReDim RouteInfo(1 To totalRP, 9) As Variant
ReDim RGarray(1 To numreceivergroups) As Variant
ReDim tnkAssign(1 To numTankerBases, 1 To totalRP) As Variant
ReDim tnkstart(1 To numTankerBases, 1 To totalRP) As Variant
ReDim tnkfinish(1 To numTankerBases, 1 To totalRP) As Variant
ReDim NewAntRoute(1 To totalRP, 1 To totalRP) As Variant
ReDim newtnkroute(1 To totalRP) As Variant
ReDim tnkcount(1 To numTankerBases) As Variant
ReDim tankersched(4, 0 To totalRP) As Variant


    numbases = 0
    numtanker = 0
    'Calculates the Earliest Start (ES) time for the first RP of a RG based on the time it takes tanker to
    'read the RP.  Also generates the amount of slack in each RG and adjusts the ES for all other RPs in route

    firstrp = 1
    For i = 2 To totalRP
        If RefuelPoints(i - 1, 1) <> RefuelPoints(i, 1) Or i = totalRP Then
            RG = RefuelPoints(i - 1, 11)    '=current RG being evaluated
            lastrp = i - 1                  '=last RP in RG
            tnkleadtime = TnkTime(tnkroutes(RG), firstrp)

            'if tanker arrival time is greater than the RG arrival, calculate the delta and store as adjustment

            If tnkleadtime > RefuelPoints(firstrp, 5) Then
                adjustment(RG) = tnkleadtime - RefuelPoints(firstrp, 5)

            Else
                adjustment(RG) = 0
            End If
            'Calculates the total RG flight time (RGtiming(i,4)) and the RG slack time (RGtiming(i,3))
            RGtiming(RG, 4) = RefuelPoints(lastrp, 5) + GreatCircleDistance(RefuelPoints(lastrp, 2), RefuelPoints(lastrp, 3),
RefuelInfo(RG, 4), RefuelInfo(RG, 5)) / MissionPlan(RG, 11)
            RGtiming(RG, 3) = RGtiming(RG, 2) - RGtiming(RG, 4) - adjustment(RG)
            firstrp = i     '=first RP in next RG

        End If

    Next i

    'adjust the ES of each RP based on the calculated adjustment delta calculated above
    For i = 1 To totalRP
        RG = RefuelPoints(i, 11)
        RefuelPoints(i, 5) = RefuelPoints(i, 5) + adjustment(RG)
    Next i

    'Populates array to keep track of which receivergroups haven't been scheduled
    For i = 1 To numreceivergroups
        RGarray(i) = 1
    Next i


    'Arranges the ant routes chronologically within their RG

    Start = 1
    Startrg = RefuelPoints(antroutes(1, 1), 11)
```

```
    prev = 1
    For i = 2 To route
       If RefuelPoints(antroutes(i, 1), 11) = Startrg And i = prev + 1 Then   'Finds the actual first route
          prev = prev + 1
       ElseIf RefuelPoints(antroutes(i, 1), 11) = Startrg Then
          Start = i 'stores location of first route in antroutes array
          GoTo getout
       End If
    Next i
getout:


       Move = 0
       For i = Start To route
          For m = 1 To totalRP
             NewAntRoute(Move + 1, m) = antroutes(i, m)              'moves route to head
          Next m
          newtnkroute(Move + 1) = tnkroutes(i)
          Move = Move + 1     'keeps track of number of routes moved to head
       Next i


    If Start <> 1 Then
       For i = 1 To Start - 1          'replaces tail
          For m = 1 To totalRP
             NewAntRoute(i + Move, m) = antroutes(i, m)
          Next m
          newtnkroute(i + Move) = tnkroutes(i)
       Next i
    End If

    'Finds the RG that is serviced by each antroute then
    'Also creates array of pertinent info for scheduling
    For i = 1 To route
       routeRG(i, 1) = RefuelPoints(NewAntRoute(i, 1), 11) 'associates a RG with each ant route
       RouteInfo(i, 1) = NewAntRoute(i, 1)
       prev = 1
       nextrp = 2
       RouteInfo(i, 3) = 0
       While NewAntRoute(i, nextrp) <> ""
          RouteInfo(i, 3) = RouteInfo(i, 3) + RPdist(nextrp, prev)    'finds total length of route
          prev = nextrp
          nextrp = nextrp + 1
       Wend
       RouteInfo(i, 2) = NewAntRoute(i, prev)

    Next i


       'Schedule receivergroups based on least slack

    For j = 1 To numreceivergroups
       Start = 1
       While RGarray(Start) = 0    'finds next unscheduled RG
          Start = Start + 1
       Wend

       'finds the unscheduled job with least slack
       leastslack = RGtiming(Start, 3)
       slackjob = Start
       For i = 1 To numreceivergroups
          If RGarray(i) = 1 Then
             If RGtiming(i, 3) < leastslack Then
                leastslack = RGtiming(i, 3)
                slackjob = i
             End If
          End If
       Next i
```

```
    RGarray(slackjob) = 0

    'This section uses LST to do final scheduling of routes


    For i = 1 To route

       If routeRG(i, 1) = slackjob Then      'for each route that is in the slack job

        'schedule tankers to routes

          Start = 1
          firstrp = RouteInfo(i, 1)       'first RP in route
          lastrp = RouteInfo(i, 2)        'last RP in route
          tnkbseused = newtnkroute(i)       'tanker base to use

          RouteInfo(i, 7) = RefuelPoints(firstrp, 5) - TnkTime(tnkbseused, firstrp)   'tanker takeoff
          RouteInfo(i, 8) = RefuelPoints(lastrp, 5) + TnkTime(tnkbseused, lastrp)     'tanker land


          'Check to see if you can use an existing tanker from base required
          For k = 0 To numbases
             If tankersched(1, k) = tnkbseused Then
                'if this tanker base already is used check for reuse possibility
                If RouteInfo(i, 7) >= tankersched(4, k) + 3 And RouteInfo(i, 8) Then 'if start is after finish of existing
                   GoTo jumpout
                Else    'shift routes backwards
                   shiftfactor = tankersched(4, k) + 3 - RouteInfo(i, 7)
                   If RGtiming(slackjob, 4) + shiftfactor <= RGtiming(slackjob, 2) Then
                      Call RPshift(slackjob, totalRP, shiftfactor)
                      RGtiming(slackjob, 4) = RGtiming(slackjob, 4) + shiftfactor
                      tankersched(1, numbases) = tnkbseused
                      tankersched(2, numbases) = tankersched(2, k)
                      tankersched(3, numbases) = RouteInfo(i, 7)
                      tankersched(4, numbases) = RouteInfo(i, 8)
                   Else
                      GoTo newbase
                   End If

                End If

             End If
          Next k



newbase:
          'otherwise assign a new tanker to service route i
          numbases = numbases + 1
          tankersched(1, numbases) = tnkbseused
          tankersched(2, numbases) = tnkcount(tnkbseused) + 1
          tankersched(3, numbases) = RouteInfo(i, 7)
          tankersched(4, numbases) = RouteInfo(i, 8)
          RouteInfo(i, 9) = numbases              'associates the column number of this tanker with the route
          tnkcount(tnkbseused) = tnkcount(tnkbseused) + 1     'update tail numbers used



          numtanker = numtanker + 1                     'update total tankers used



       End If
jumpout:
     Next i

  Next j
```

```
End Sub

Sub RPshift(RG, totalRP, shift)

For i = 1 To totalRP
   If RefuelPoints(i, 11) = RG Then
      RefuelPoints(i, 5) = RefuelPoints(i, 5) + shift

   End If
Next i

End Sub

Function fighterescort(k, totalRP)
'This function is called to find the "Escort" point for a fighter's refueling point

   temp = k + 1
   fighterescort = 0
      'Looks at next RP.  If it belongs to same RG and has different escort point then save this RP
      'Otherwise look at the next RP
   If temp <= totalRP Then
      While RefuelPoints(k, 1) = RefuelPoints(temp, 1) And fighterescort = 0
         If RefuelPoints(k, 8) = RefuelPoints(temp, 8) Then
            temp = temp + 1     'Increments temp
            If temp > totalRP Then GoTo final

         Else
            fighterescort = RefuelPoints(temp, 8)
         End If

      Wend
   End If
final:
End Function

Function tankerFind(k, rpcount)
' This subroutine determines which tanker should be selected to service the first RP in a route

   bestprob = -999
   For i = 1 To numTankerBases
   'Checks to see if tanker can make roundtrip journey and meet offload requirement
   'Roundtrip fuel < max capacity - fuel reserve - required offload
   'If it is within range, calculate the ACO prob function for it.
      If TankerFuel(i, k) <= TnkParams(i, 3) - RefuelPoints(k, 4) Then

         probfunction = pheromone(k, rpcount + i) ^ alpha + (10000 / (RPdist(k, rpcount + i) + 1)) ^ beta

      End If
      If probfunction > bestprob Then
         bestprob = probfunction
         tankerFind = i  'stores tankerbase if probfunction is better than any previous
      End If

   Next i

End Function
```

```
'Subroutine Name:   TankerReturn()
'Functionality:     This button checks the user input for errors.  It insures that the base listing and
'                   tanker type are valid as well as the number of tankers at each base.  If all data is
'                   valid then it returns user to the main menu

'Arguments:         None

Sub TankerReturn()

    ' Reads in tanker base sheet for error checking

    Set tempfile = Sheets("TankerBases").Range("A4").CurrentRegion
    tankerBases = tempfile.Offset(1, 0).Resize(tempfile.Rows.Count - 1, tempfile.Columns.Count)
    numTankerBases = tempfile.Rows.Count - 1

    'Reads in AIRBASES worksheet to check for correct ICAO

    Set tempfile = Sheets("AIRBASES").Range("A3").CurrentRegion
    globalbaselist = tempfile.Offset(1, 0).Resize(tempfile.Rows.Count - 1, tempfile.Columns.Count)
    NumGlobalBases = tempfile.Rows.Count - 1

    'Reads in TANKER DATA worksheet to check for correct tanker type

    Set tempfile = Sheets("TANKER DATA").Range("A4").CurrentRegion
    tankers = tempfile.Offset(1, 0).Resize(tempfile.Rows.Count - 1, tempfile.Columns.Count)
    numTankertypes = tempfile.Rows.Count - 1

    ' Checks to see if the entered tanker base is already present in the Airbase listing.  If not print
    ' out an error message and return control to that sheet to allow user to make required changes

    For i = 1 To numTankerBases
       RowIndex = Find(tankerBases(i, 1), globalbaselist, NumGlobalBases)
       If RowIndex = -1 Then
          MsgBox ("Error: " & tankerBases(i, 1) & " in row " & i + 3 & " is not a valid ICAO.  Either enter a new ICAO or add this
ICAO to the Airbase Listing")
          GoTo Quit
       End If

    ' Checks to see if user enters a positive number of tanker bases.  If not print out an error message
    ' and return control to that sheet to allow user to make required changes

       If tankerBases(i, 2) <= 0 Then
          MsgBox ("Error: the number of tankers entered in row " & i + 3 & " is not valid. Must enter a positive number of planes")
          GoTo Quit
       End If

    ' Checks to see if the entered tanker type is already present in the Tanker listing.  If not print
    ' out an error message and return control to that sheet to allow user to make required changes

       RowIndex = Find(tankerBases(i, 3), tankers, numTankertypes)
       If RowIndex = -1 Then
          MsgBox ("Error: " & tankerBases(i, 3) & " in row " & i + 3 & " is not a valid tanker type.  Either enter a new tanker or add this
tanker to the Tanker Listing")
          GoTo Quit
       End If
    Next i

    ' If all entered data is correct, return to main menu

    Sheets("Menu").Activate

     ' If a piece of data is incorrect, then return to TANKERBASES worksheet so corrections can be made
Quit:
End Sub

'Subroutine Name:   AirbaseReturn
'Functionality:     This button checks the user input for errors.  It insures that the base listing and
```

```
'                tanker type are valid as well as the number of tankers at each base.  If all data is
'                valid then it returns user to the previous worksheet.
'Arguments:         None

Sub AirbaseReturn()

   'Reads in AIRBASES worksheet to check for correct ICAO

   Set tempfile = Range("A3").CurrentRegion
   globalbaselist = tempfile.Offset(1, 0).Resize(tempfile.Rows.Count - 1, tempfile.Columns.Count)
   NumGlobalBases = tempfile.Rows.Count - 1

   ' Checks to see if the entered latitude and longitude are valid.  FIrst check is to ensure a value is
   ' entered.  Secondly it ensures lat/long value is less than 180 degrees.  If invalid data is entered
   ' the user is notified the airbase and row that is invalid and allows user to update.

   For i = 1 To NumGlobalBases
      If globalbaselist(i, 4) = "" Or Abs(globalbaselist(i, 4)) > 18000 Then
         MsgBox ("Error:  Need to enter a valid latitude for airbase " & globalbaselist(i, 1) & " in row " & i + 3)
         GoTo Quit
      End If

       If globalbaselist(i, 5) = "" Or Abs(globalbaselist(i, 5)) > 18000 Then
         MsgBox ("Error:  Need to enter a valid longitude for airbase " & globalbaselist(i, 1) & " in row " & i + 3)
         GoTo Quit
      End If

   Next i

   ' If all the information is correct, then return to the previous screen

   Sheets(AirbaseCall).Activate         'returns to main menu if called from it


   ' If a piece of data is incorrect, then return to Airbases worksheet so corrections can be made
Quit:

End Sub

'Subroutine Name:   ReceiverGroupReturn
'Functionality:     This button checks the user input for errors.  It insures that the base listings and
'                receiver types are valid as well as the number of tankers at each base.  If all data is
'                valid then it returns user to the previous worksheet.
'Arguments:         None

Sub ReceiverGroupReturn()

   'Reads in INPUT worksheet for error checking

   Set tempfile = Sheets("INPUT").Range("A6").CurrentRegion
   Receiverdata = tempfile.Offset(1, 0).Resize(tempfile.Rows.Count - 1, tempfile.Columns.Count)
   numreceivergroups = tempfile.Rows.Count - 1

   'Reads in AIRBASES worksheet to check for correct ICAO

   Set tempfile = Sheets("AIRBASES").Range("A3").CurrentRegion
   globalbaselist = tempfile.Offset(1, 0).Resize(tempfile.Rows.Count - 1, tempfile.Columns.Count)
   NumGlobalBases = tempfile.Rows.Count - 1

   'Reads in RECEIVER DATA information for error checking

   Set tempfile = Sheets("RECEIVER DATA").Range("A5").CurrentRegion
   receivers = tempfile.Offset(2, 0).Resize(tempfile.Rows.Count - 2, tempfile.Columns.Count)
   Numreceivertypes = tempfile.Rows.Count - 2

   ' Checks to see if ICAO listings for bases are currently defined
   For i = 1 To numreceivergroups
```

```
        'Check Receiver type
        RowIndex = Find(Receiverdata(i, 2), receivers, Numreceivertypes)
        If RowIndex = -1 Then
            MsgBox ("Error: " & Receiverdata(i, 2) & " in row " & i + 6 & " is not a valid receiver type.  Either enter a new receiver or
add this receiver to the Receiver Data Listing")
            GoTo Quit
        End If

        'Check Base of Origin
        RowIndex = Find(Receiverdata(i, 4), globalbaselist, NumGlobalBases)
        If RowIndex = -1 Then
            MsgBox ("Error: " & Receiverdata(i, 4) & " in row " & i + 6 & " is not a valid ICAO.  Either enter a new ICAO or add this
ICAO to the Airbase Listing")
            GoTo Quit
        End If

        'Check destination base
        RowIndex = Find(Receiverdata(i, 5), globalbaselist, NumGlobalBases)
        If RowIndex = -1 Then
            MsgBox ("Error: " & Receiverdata(i, 5) & " in row " & i + 6 & " is not a valid ICAO.  Either enter a new ICAO or add this
ICAO to the Airbase Listing")
            GoTo Quit
        End If

        ' Checks to see if user enters a positive number of tanker bases.  If not print out an error message
        ' and return control to that sheet to allow user to make required changes

        If Receiverdata(i, 3) <= 0 Then
            MsgBox ("Error: the number of receivers entered in row " & i + 6 & " is not valid. Must enter a positive number of planes")
            GoTo Quit
        End If

        ' Checks to see if user enters a positive for ALD and RDD, and that ALD is before RDD.  If not print out an error message
        ' and return control to that sheet to allow user to make required changes

        If Receiverdata(i, 12) <= 0 Then
            MsgBox ("Error: the ALD in row " & i + 6 & " is not valid. Must enter a positive number of planes")
            GoTo Quit
        End If

        If Receiverdata(i, 13) <= 0 Then
            MsgBox ("Error: the RDD in row " & i + 6 & " is not valid. Must enter a positive number of planes")
            GoTo Quit
        End If

        If Receiverdata(i, 12) >= Receiverdata(i, 13) Then
            MsgBox ("Error: ALD must be before RDD in row " & i + 6)
            GoTo Quit
        End If


    Next i

    Sheets("Menu").Activate
Quit:

End Sub

'Subroutine Name:   ReceiverReturn
'Functionality:     This button returns user to the previous worksheet.
'Arguments:         None
Sub ReceiverReturn()

    ' Return from Receiver Data Worksheet

    Sheets(ReceiverCall).Activate        'returns to main menu if called from it
```

```
End Sub
'Subroutine Name:   TankerDataReturn
'Functionality:     This button returns user to the previous worksheet.
'Arguments:         None
Sub TankerDataReturn()

    ' Return from Receiver Data Worksheet

    Sheets(TankerCall).Activate         'returns to main menu if called from it


End Sub

Public AirbaseCall As String
Public TankerCall As String
Public ReceiverCall As String


'Subroutine Name:   InputReceiverGroups()
'Functionality:     This function takes user to sheet to add or change
'                   receiver groups

'Arguments:         None

Sub InputReceiverGroups()

    Sheets("INPUT").Activate

End Sub

'Subroutine Name:   MainMenu
'Functionality:     This function returns user to main menu
'Arguments:         None

Sub MainMenu()

    Sheets("Menu").Activate

End Sub

'Subroutine Name:   TankerInput
'Functionality:     This function allows user to input tanker bases available
'Arguments:         None

Sub TankerInput()

    Sheets("TankerBases").Activate

End Sub

'Subroutine Name:   ReceiverAircraftData
'Functionality:     This function allows user to add or change receiver aircraft
'                   performance data
'Arguments:         None

Sub ReceiverAircraftData()
    ReceiverCall = ActiveCell.Worksheet.name
    Sheets("RECEIVER DATA").Activate

End Sub

'Subroutine Name:   AirbaseInfo
'Functionality:     This function allows user to adjust worldwide airbase info
'Arguments:         None

Sub AirbaseInfo()
    AirbaseCall = ActiveCell.Worksheet.name
    Sheets("AIRBASES").Activate
```

```
End Sub

'Subroutine Name:   TankerAircraftData
'Functionality:     This function allows user to add or change tanker performance data
'Arguments:         None

Sub TankerAircraftData()
   TankerCall = ActiveCell.Worksheet.name
   Sheets("TANKER DATA").Activate

End Sub

'Subroutine Name:   RefuelingPoints
'Functionality:     This function allows user to view refueling point info
'Arguments:         None

Sub RefuelingPoints()

   Sheets("Refueling Points").Activate

End Sub

'Subroutine Name:   MissionPlan
'Functionality:     This function allows user to view final mission plan info

Sub MissionPlan()

   Sheets("Best Mission Plan").Activate

End Sub


'Subroutine Name:   ClearTankerBases()
'Functionality:     This function clears the Tanker Base Worksheet

'Arguments:         None

Sub ClearTankerBases()
   Set tempfile = Range("A4").CurrentRegion
   numTankerBases = tempfile.Rows.Count - 1
   Range(Cells(4, 1), Cells(numTankerBases + 4, 3)).Select
   Selection.ClearContents
End Sub

'Subroutine Name:   ClearReceiverGroups
'Functionality:     This function clears the Receiver Group Worksheet
'Arguments:         None

Sub ClearReceiverGroups()
   Set tempfile = Sheets("INPUT").Range("A6").CurrentRegion
   numreceivergroups = tempfile.Rows.Count - 1
   Range(Cells(7, 1), Cells(numreceivergroups + 7, 23)).Select
   Selection.ClearContents
End Sub
```

```
'Function Name: Ceiling
'Functionality: This function rounds fractional numbers
'              to the next highest integer
'Arguments:     x - The function finds the ceiling of x
'Return Value:  Ceiling - the ceiling of x

Function Ceiling(x)
   temp = CInt(x)
   temp1 = temp - x
   If temp1 > 0 Then
      Ceiling = temp
   Else
      Ceiling = temp + 1
   End If
End Function

'Function Name: Max
'Functionality: This function finds the largest of two numbers
'Arguments:     a  and b
'Return Value:  Max - the largest of the two arguments a and b

Function Max(a, b)
   If a > b Then
      Max = a
   Else
      Max = b
   End If
End Function

'Function Name: Floor
'Functionality: This function rounds fractional numbers
'              to the next lolwest integer
'Arguments:     x - The function finds the floor of x
'Return Value:  floor - the floor of x

Function Floor(x)

   temp = Abs(x)
   temp = Ceiling(temp)
   If x > 0 Then
      Floor = temp - 1
   Else
      Floor = -temp
   End If
End Function

'Function Name: Find
'Functionality: This function determines the row position
'              of a given aircraft in the distcalc or
'              aircraft performance matrices
'Arguments:     item - the name of the aircraft to find
'              list - the matrix to search
'              total - the number of rows in the search matrix
'Return Value:  pos - the row position of the aircraft

Function Find(item, list, total)
   pos = 1
   found = False
      While Not found
         If StrComp(item, list(pos, 1)) = 0 Then
            Find = pos
            found = True
         Else
            pos = pos + 1
         End If
         If pos > total Then
            Find = -1
```

```
            found = True
          End If
        Wend
End Function

'Function Name: OffloadCheck
'Functionality: This function determines the minimum number of
'               tankers that can service a given refueling point
'Arguments:     offload - the required fuel at refueling point
'               latRP - latitude of the refueling point
'               longRP - longitude of the refueling point
'Return Value:  pos - the minimum number of tankers that can meet the offload requirement
Function OffloadCheck(offload, latRP, longRP)

    mintankers = 100    'Initialize number of tankers needed to be very large
    For t = 1 To numTankerBases
    ' Calculates distance between RP and tankerbase, finds the tanker type, and then calculates the roundtrip fuel
    ' needed to reach refueling point and then return (Stores the fuel needed as FuelTravel)
       dist = GreatCircleDistance(latRP, longRP, tankBases(t, 3), tankBases(t, 4))
       RowIndex = Find(tankBases(t, 6), tankers, numTankertypes)
       FuelTravel = tankers(RowIndex, 7) + (2 * dist - 100) * tankers(RowIndex, 4) / tankers(RowIndex, 3)
       'Calculate the offload available at the RP
       maxtankeroffload = tankers(RowIndex, 5) - tankers(RowIndex, 6) - FuelTravel
       'If tanker can refuel, calculate the number needed to meet this RP requirement
       If maxtankeroffload > 0 Then required = Ceiling(offload / maxtankeroffload)
       If required < mintankers Then mintankers = required
    Next t
       offload = mintankers    'Assigns offload to the minimum number of tankers required to refuel this RP

End Function


'Subroutine Name:   setWddefaults
'Functionality:     This function sets the winds to zero
'Arguments:         None

Sub setWddefaults()

missions = Application.Count(Sheets("INPUT").Range("C7:C200"))

For i = 1 To missions
    Sheets("WINDS").Cells(i + 6, 1) = 0
    Sheets("WINDS").Cells(i + 6, 2) = 0
Next i

End Sub

'Subroutine Name:   setGWdefaults()
'Functionality:     This function sets the default values for the weight
'                   characteristics of receiver aircraft.  Including:
'                      1)  Minimum Weight Empty
'                      2)  Payload Weight
'                      3)  Fuel Weight
'                   The default values are set in the AIRCRAFT PERFORMANCE sheet
'Arguments:         None

Sub setdefaults()

missions = Application.Count(Sheets("INPUT").Range("C7:C207"))
rdata = Sheets("INPUT").Range("A7:P207")

rec = Application.Count(Sheets("AIRCRAFT PERFORMANCE").Range("B5:B37"))
receivers = Sheets("AIRCRAFT PERFORMANCE").Range("A5:P37")

For i = 1 To missions
    ind = Find(rdata(i, 2), receivers, rec)
    Sheets("INPUT").Cells(i + 6, 9) = receivers(ind, 10)
    Sheets("INPUT").Cells(i + 6, 10) = receivers(ind, 11)
```

```
    Sheets("INPUT").Cells(i + 6, 11) = receivers(ind, 5)
    Sheets("INPUT").Cells(i + 6, 8) = receivers(ind, 8)
    Sheets("INPUT").Cells(i + 6, 7) = receivers(ind, 3)
Next i
End Sub
```

```
'Module:    DistCalc
'Function:  This module contains the functions for calculating the great circle
'          distance from the origin to the destination bases


'Function Name: DecDeg
'Functionality: To decode the DDDMM.M format (where D=degrees, M=Minutes) for Latitude
'              and Longitude to degrees.
'Arguments:     Number - Value passed to function in DDDMM.M format
'Return Value:  Temp - the Latitude or Longitude in degrees
Private Function DecDeg(number1)

   num = Abs(number1)   ' Get absolute value of Number to use in Int()

   temp = Int(num / 100) + (num / 100 - Int(num / 100)) / 0.6
                        ' Convert by separating integer degrees from
                        '  minutes portion.  Then divide minutes by 60
                        '  to get fractional degrees and add to integer
                        '  degrees.

   If num > number1 Then           ' Check that Temp has same sign (+/-) as Number
      temp = -temp              '  before assigning to return value
   End If
   DecDeg = temp                ' Assign Temp to function's return value
End Function
Function DegDec(number2)

   num = Abs(number2)

   temp = (Floor(num) * 100) + (60 * (num - Floor(num)))

   If number2 < 0 Then temp = -temp

   DegDec = temp
End Function

'Function Name: GreatCircleDistance
'Functionality: To compute great circle distance between two points on Earth.  Points
'              are (Latitude1, Longitude1) and (Latitude2, Longitude2).  This function
'              accepts latitude and longitude in real degrees or in DDDMM.M format.
'Arguments:     latitude1 - origin latitude
'              longitude1 - orgin longitude
'              latitude2 - destination latitude
'              longitude2 - destination longitude
'Return Value:  GreatCircleDistance - the great circle distance

Function GreatCircleDistance(latitude1, longitude1, latitude2, longitude2)

   Deg2Rad = 3.14159265358979 / 180    'Define constants
   Rad2Deg = 180 / 3.14159265358979    'for angle conversions
   NMperDeg = 60

   lat1 = latitude1
   lat2 = latitude2
   long1 = longitude1
   long2 = longitude2

   If (Abs(lat1) > 90) Or (Abs(lat2) > 90) Or (Abs(long1) > 180) Or (Abs(long2) > 180) Then
      lat1 = DecDeg(lat1)   ' Assumes all coordinates are in same
      lat2 = DecDeg(lat2)   '  format. If any are found in DDDMM.M
      long1 = DecDeg(long1) '  format then convert all to degrees.
      long2 = DecDeg(long2)
   End If

   lat1 = lat1 * Deg2Rad     ' Convert all degrees to radians
   lat2 = lat2 * Deg2Rad
   long1 = long1 * Deg2Rad
```

```
    long2 = long2 * Deg2Rad

    temp = Cos(lat1) * Cos(lat2) * Cos(long2 - long1)

    If lat1 = lat2 And long1 = long2 Then
       GreatCircleDistance = NMperDeg * temp
    Else
       temp = Application.Acos(temp + Sin(lat1) * Sin(lat2)) * Rad2Deg
                              ' Calculated the angle of the great circle
                              '  arc between the two points.  Formula
                              '  came from original AMCSAF Distcalc
                              '  spreadsheet.  Uses Excel's ACOS().

       GreatCircleDistance = NMperDeg * temp  ' Convert arc degrees to NM and return
    End If
End Function
Function getAz(latitude1, longitude1, latitude2, longitude2)

    Deg2Rad = 3.14159265358979 / 180    'Define constants
    Rad2Deg = 180 / 3.14159265358979    'for angle conversions
    NMperDeg = 60

    lat1 = latitude1
    long1 = longitude1
    lat2 = latitude2
    long2 = longitude2
    dist = GreatCircleDistance(lat1, long1, lat2, long2)

    If (Abs(lat2) > 90) Or (Abs(long1) > 180) Or (Abs(long2) > 180) Then
       lat1 = DecDeg(lat1)   ' Assumes all coordinates are in same
       lat2 = DecDeg(lat2)   '  format. If any are found in DDDMM.M
       long1 = DecDeg(long1) '  format then convert all to degrees.
       long2 = DecDeg(long2)
    End If

    lat1 = lat1 * Deg2Rad
    lat2 = lat2 * Deg2Rad
    long1 = long1 * Deg2Rad
    long2 = long2 * Deg2Rad
    dist = dist / NMperDeg
    dist = dist * Deg2Rad

    sinAz = (Cos(lat2) * Sin(long2 - long1) / Sin(dist))
    cosAz = ((Sin(lat2) - (Cos(dist) * Sin(lat1))) / (Sin(dist) * Cos(lat1)))
    If sinAz >= 0 And cosAz >= 0 Then
       temp = Application.Asin(sinAz)
    ElseIf sinAz >= 0 And cosAz < 0 Then
       temp = 3.14159265358979 - Application.Asin(sinAz)
    ElseIf cosAz >= 0 Then
       temp = -Application.Acos(cosAz)
    Else
       temp = -(3.14159265358979 + Application.Asin(sinAz))
    End If
    temp = temp * Rad2Deg
    getAz = temp
End Function
Function getLat(latitude1, distance, azymuth)

    Deg2Rad = 3.14159265358979 / 180    'Define constants
    Rad2Deg = 180 / 3.14159265358979    'for angle conversions
    NMperDeg = 60

    lat1 = latitude1
    dist = distance
    Az = azymuth

    If (Abs(lat1) > 90) Then lat1 = DecDeg(lat1)
```

```
    lat1 = lat1 * Deg2Rad
    dist = dist / NMperDeg
    dist = dist * Deg2Rad
    Az = Az * Deg2Rad

    temp = Application.Acos(Sin(lat1) * Cos(dist) + Cos(lat1) * Sin(dist) * Cos(Az))
    temp = temp * Rad2Deg
    temp = 90 - temp
    getLat = DegDec(temp)
End Function
Function getLong(longitude1, distance, azymuth, latitudeRP)

    Deg2Rad = 3.14159265358979 / 180    'Define constants
    Rad2Deg = 180 / 3.14159265358979    'for angle conversions
    NMperDeg = 60

    long1 = longitude1
    dist = distance
    Az = azymuth
    latRP = latitudeRP

    If (Abs(long1) > 90) Or (Abs(latRP) > 90) Then
       long1 = DecDeg(long1)
       latRP = DecDeg(latRP)
    End If

    dist = dist / NMperDeg
    dist = dist * Deg2Rad
    Az = Az * Deg2Rad
    latRP = latRP * Deg2Rad
    long1 = long1 * Deg2Rad

    temp = Application.Asin(Sin(dist) * Sin(Az) / Cos(latRP))
    temp = temp + long1
    temp = temp * Rad2Deg
    If temp > 180 Then temp = temp - 360
    getLong = DegDec(temp)
End Function




Function TrueCourse(dist, latitude1, longitude1, latitude2, longitude2)

    Deg2Rad = 3.14159265358979 / 180    'Define constants
    Rad2Deg = 180 / 3.14159265358979    'for angle conversions
    p = 3.1415926535897

    la1 = latitude1
    lg1 = longitude1
    la2 = latitude2
    lg2 = longitude2

    If (Abs(la1) > 90) Or (Abs(la2) > 90) Or (Abs(lg1) > 180) Or (Abs(lg2) > 180) Then
       la1 = DecDeg(la1)   ' Assumes all coordinates are in same
       la2 = DecDeg(la2)   '  format. If any are found in DDDMM.M
       lg1 = DecDeg(lg1) '  format then convert all to degrees.
       lg2 = DecDeg(lg2)
    End If

    la1 = la1 * Deg2Rad     ' Convert all degrees to radians
    la2 = la2 * Deg2Rad
    lg1 = lg1 * Deg2Rad
    lg2 = lg2 * Deg2Rad
    d = (dist / 60) * Deg2Rad
```

```
    H1 = Application.Acos((Sin(la2) - Sin(la1) * Cos(d)) / (Sin(d) * Cos(la1)))
    H2 = Application.Acos((Sin(la1) - Sin(la2) * Cos(d)) / (Sin(d) * Cos(la2)))

    If Sin(lg2 - lg1) < 0 Then
       Hi1 = H1
    Else
       Hi1 = 2 * p - H1
    End If

    If Sin(lg1 - lg2) < 0 Then
       Hi2 = H2
    Else
       Hi2 = 2 * p - H2
    End If

    If Hi2 >= p Then
       Hi2 = Hi2 - p
    Else
       Hi2 = Hi2 + p
    End If
    TrueCourse = (Hi1 + Hi2) / 2 * Rad2Deg

End Function

Function GroundSpeed(tas, TC, Wd, Wv)

    Deg2Rad = 3.14159265358979 / 180    'Define constants
    Rad2Deg = 180 / 3.14159265358979    'for angle conversions

    TCr = TC * Deg2Rad

    Wdr = Wd * Deg2Rad
    DCA = Application.Asin((Wv / tas) * Sin(Wdr - TCr))
    GroundSpeed = tas * Cos(DCA) - Wv * Cos(Wdr - TCr)

End Function
```

```
'Function Name: fuelburn
'Functionality: This function is used to determine the fuel
'               burned by a given fighter for a period of
'               flight given by: Flight time = Distance/True Air Speed
'               The algorithm assumes a nominal flight altitude and
'               true air speed. The fuel flow is calculated with a third
'               order polynomial model of the fuel flow depending on gross weight.
'               It is assumed that the fighter's fuel is burned down to
'               the fuel reserve level and then completely refueled.
'Arguments:     dist - the distance the fighter will travel
'               tas - the true airspeed the fighter will travel at
'               r - the fighter performance matrix
'               j - the position of the desired fighter in the performance matrix
'Return Value:  totfb - total fuel burned over the flight

Function recflburn(dist, rate, fuelcap, reserve, minwt, cargo, climbf, c1, c2, c3, c4)

    fb = 0
    totfb = climbf
    ff = 0
    mult = 0

    gw = 0
    gwi = fuelcap + minwt + cargo
    maxburn = fuelcap - reserve

    Flighttime = dist / rate
    dt = 0.01

    For t = 1 To Flighttime * 100
      Nar = Ceiling(totfb / maxburn) - 1
      gw = gwi - totfb + Nar * maxburn
      ff = c1 + c2 * gw + c3 * gw * gw + c4 * gw * gw * gw
      fb = ff * dt
      totfb = totfb + fb
    Next t

    recflburn = totfb

End Function

Function tnkrflburn(dist, rrate, trate, rgs, tgs, fuelcap, reserve, minwt, cargo, climbf, ralt, talt, c1, c2, c3, c4, c5, c6, c7, test)

    t = 0
    dt = 0.01
    maxburn = fuelcap - reserve
    fb = 0
    ff = 0
    gw = 0
    gwi = fuelcap + minwt + cargo
    totfb = climbf

    If StrComp(test, "F") = 0 Then
       fltm = 0.5 * dist / rgs + 0.5 * dist / tgs
    ElseIf StrComp(test, "R") = 0 Then
       fltm = dist / rgs
    Else
       fltm = dist / tgs
    End If

    While t < fltm * 100
       If StrComp(test, "F") = 0 Then
          If t * dt < 0.5 * fltm Then
             alt = ralt
             tas = rrate
          Else
             alt = talt
```

```
            tas = trate
        End If
    Else
        alt = talt
        tas = trate
    End If

        gw = gwi - totfb
        fflow = c1 + c2 * alt + c3 * alt * alt + c4 * tas + c5 * tas * tas + c6 * gw + c7 * gw * gw
        fb = fflow * dt
        totfb = totfb + fb

        t = t + 1

  Wend
  tnkrflburn = totfb

End Function
```

Bibliography

"Aerospace Weapons", *Airman*, January 2002, pp 44-56.

Batitti, Roberto and G. Tecchiolli. "The Reactive Tabu Search", *ORSA Journal on Computing*, Vol 6, No 2, Spring 1994, pp 126-140.

Bonabeua, Eric and Guy Theraulaz. "Swarm Smarts", *Scientific American*, Vol 282, No 3, March 2000, pp 72-79.

Bullnheimer, Bernd, Richard Hartl, and Christine Strauss. "Applying the Ant System to the Vehicle Routing Problem", 2nd International Conference on Metaheuristics, Sophia-Antipolis, France, July 21-24, 1997.

Bullnheimer, Bernd, Richard Hartl, and Christine Strauss. "An Improved Ant System Algorithm for the Vehicle Routing Problem", *Annals of Operations Research*, Vol 89, 1999b, pp 319-328.

Calhoun, Kevin M. *A Tabu Search for Scheduling and Rescheduling Combat Aircraft*. MS Thesis. AFIT/GOR/ENS/00M-06, School of Engineering and Management, Air Force Institute of Technology, Wright-Patterson AFB OH, March 2000.

Capehart, Shay R. *A Tabu Search Metaheuristic for the Air Refueling Tanker Assignment Problem*. MS Thesis. AFIT/GOR/ENS/00M-07, School of Engineering and Management, Air Force Institute of Technology, Wright-Patterson AFB OH, March 2000.

Cordeau, Jean-Francois, M. Gendreau, and G. Laporte. "A Tabu Search Heuristic for Periodic and MultiDepot Vehicle Routing Problems", *Networks*, 1997, pp 105-119.

Corne, David and others. New Ideas in Optimization. London: McGraw-Hill, 1999.

Dorigo, Marco and Gianni Di Caro. "Ant Colony Optimization: A New Metaheuristic", Proceeding of the Congress on Evolutionary Computation, Washington DC, July 6-9, 1999.

Glover, Fred and M. Laguna. Tabu Search. Boston: Kluwer Academic Publishers, 1997.

Hostler, Harry C. *Air Refueling Tanker Scheduling*. MS Thesis. AFIT/GST/ENS/87M-7, School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB OH, March 1987.

Iannuzzi, Philip A. "50 Years Without Air Refueling Doctrine", *Airlift/Tanker Quarterly*, Vol 5, No 2, Spring 1997.

Iannuzzi, Philip A. "50 Years Without Air Refueling Doctrine – Part II", *Airlift/Tanker Quarterly*, Vol 5, No 3, Summer 1997, pp. 15-21.

Logicon. "Combined Mating and Ranging Planning System Overview." Slide presentation, Information Technology Group, Logicon Inc., 1996.

Morton, Thomas E. and David W. Pentico. Heuristic Scheduling Systems. New York: John Wiley & Sons, Inc, 1993.

Newman, Richard J. "Tankers and Lifters for a Distant War", *Air Force Magazine*, Vol 85, No 1, January 2002, pp 56-60.

Pinedo, Michael. Scheduling: Theory, Algorithms, and Systems. Englewood Cliffs NJ: Prentice Hall, 1995.

Silver, Edward A., Victor Vidal, and Dominique de Werra. "A Tutorial on Heuristic Methods", *European Journal of Operational Research*, Vol 5, 1980, pp 153-162.

Simpson, Richard. "Command of Theater Air Mobility Forces During the Air War Over Serbia", *Airlift/Tanker Quarterly*, Volume 8, Number 3, Summer 2000, pp 10-13.

Tekelioglu, Umit H. *A Reactive Tabu Search Metaheuristic Extension of the Air Refueling Tanker Assignment Problem*. MS Thesis. AFIT/GOR/ENS/01M-16. School of Engineering and Management, Air Force Institute of Technology, Wright-Patterson AFB OH, March 2001.

"The AMC Barrel Allocator: Technical Overview", Excerpt from unpublished article. n. pag. http://www-2.cs.cmu/edu/afs/cs/project/ozone/www/barrel/tech-oview/barrel-oview.html, 6 February 2002.

"USAF Almanac: Gallery of USAF Weapons", *Air Force Magazine*, Vol 84, No5, May 2001, pp. 132-158.

Wiley, Victor D. *The Aerial Fleet Refueling Problem*. PhD Dissertation. University of Texas at Austin, Austin TX, 2001.

Winston, Wayne L. Operations Research Applications and Algorithms. Belmont CA: Duxbury Press, 1994.

Zanakis, Stelios H. and James R. Evans. "Heuristic Optimization: Why, When and How to Use It", *Interfaces*, Vol 11, No 5, October 1981, pp 84-91.

Vita

Captain RonJon Annaballi graduated from Coatesville Area Senior High School in Coatesville, Pennsylvania in June 1989. He accepted an Air Force ROTC scholarship to attend Lehigh University in Bethlehem, Pennsylvania where he graduated with a Bachelor of Science degree in Computer Engineering in May 1993.

His first assignment was at Wright-Patterson Air Force Base, Ohio in September 1993. He served as a Software Controls Engineer, and later Executive Officer in the Materials Directorate of Wright Laboratory. In January 1997, he was assigned to the 31st Test and Evaluation Squadron, Detachment 1 at Kirtland Air Force Base, New Mexico. At Kirtland, he served as a Military Utility Analyst before serving as the Chief of the Unmanned Aerial Vehicle Branch and later as the Chief of the Advanced Combat Aircraft Branch.

In August 2000, he entered the Graduate School of Engineering and Management, Air Force Institute of Technology at Wright-Patterson AFB, OH. Upon graduation, Captain Annaballi will be assigned to the United States Strategic Command at Offutt Air Force Base, Nebraska.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
11-03-2002	Master's Thesis	July 2001 - March 2002

4. TITLE AND SUBTITLE

A MULTIPLE ANT COLONY OPTIMIZATION METAHEURISTIC FOR THE AIR REFUELING TANKER ASSIGNMENT PROBLEM

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Annaballi, RonJon, Captain, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 P Street, Building 640
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GOR/ENS/02-01

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

HQ AMC/XPY
402 Scott Drive, Unit 3L3
Scott AFB, IL 62225-5307
DSN: 576-5954
Major Dave Ryer

Air Force Office of Scientific Research
801 N. Randolph St, Room 933
Arlington, VA 22203-1977
(703) 696-8431
Major Juan Vasquez

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

A key tenet to the Air Force's vision of Global Vigilance, Reach, and Power is the ability to project power via the use of aerial refueling. Scheduling of limited tanker resources is a major concern for Air Mobility Command (AMC). Currently the Combined Mating and Ranging Planning System (CMARPS) is used to plan aerial refueling operations, however due to the complex nature of the program and the length of time needed to run a scenario, the need for a simple tool that runs in much shorter time is desired.

Ant colony algorithms are recently developed heuristics for finding solutions to difficult optimization problems based on simulation the foraging behavior of ant colonies. It is a a distributive metaheuristic that combines an adaptive memory function with a local heuristic funtion to repeatedly construct possible solutions which can then be evaluated. Using multiple ant colony heuristics combined with a simple scheduling algorithm and modeling the Tanker Assignment Problem as a modified Multiple Depot Vehicle Routing Problem, an Excel based spreadsheet tool was developed which generates very good solutions in very short time.

15. SUBJECT TERMS

Ant Colony optimization, Heuristics, Metaheuristics, Tanker Scheduling, vehicle routing problem, multiple depot vehicle routing problem

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. James T. Moore, AFIT/ENS
U	U	U	UU	98	19b. TELEPHONE NUMBER *(Include area code)* (937) 255-6565, ext 4337 (james.moore@afit.edu)