3-9-2009

An Advanced Tabu Search Approach to Solving the Mixed Payload Airlift Load Planning Problem

Robert Larry Nance

An Advanced Tabu Search Approach to Solving the Mixed Payload Airlift Load Planning Problem

THESIS

R. Larry Nance, Maj, USAF
AFIT/GOR/ENS/09-11

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GOR/ENS/09-11

An Advanced Tabu Search Approach to Solving the Mixed Payload Airlift Load Planning 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

R. Larry Nance, BS
Major, USAF

March 2009

AFIT/GOR/ENS/09-11

An Advanced Tabu Search Approach to Solving the Mixed Payload Airlift Load Planning Problem

R. Larry Nance, BS
Major, USAF

Approved:

_____	_____
August G. Roesener (Chairman)	Date
_____	_____
James T. Moore (Co-Chairman)	Date

Abstract

Military airlift is vital to any nation's ability to project power on a global scale. In the United States, the vast majority of airlift responsibility lies with the Air Force's Air Mobility Command (AMC). As with most military endeavors of this magnitude, airlift comes at a great financial cost; it is therefore imperative to utilize the Air Force's airlift fleet in an efficient manner. One aspect of efficiency involves transporting a set of cargo items using the fewest number of aircraft possible. This is called the Mixed Payload Airlift Load Planning (MPALP) problem. This paper presents a new tabu search based two-dimensional bin packing algorithm which produces high quality solutions to the MPALP using C-5 and C-17 aircraft. This algorithm, called the Mixed Payload Airlift Load Planning Tabu Search (MPALPTS), surpasses previous research conducted in this area because, in addition to pure pallet cargo loads, MPALPTS can accommodate rolling stock cargo (i.e. tanks, trucks, HMMMVs, etc.) while still maintaining aircraft feasibility with respect to aircraft center of balance, mandatory cargo separations, aircraft floor structural limitations, etc. Furthermore, while this research is currently restricted to C-5 and C-17 aircraft, MPALPTS is capable of modeling nearly any type of cargo aircraft and requires a limited number of assumptions thereby making it applicable to operational missions. To demonstrate its effectiveness, the load plans generated by MPALPTS are directly compared to those generated by the Automated Air Load Planning Software (AALPS) for a given cargo set; AALPS is the load planning software currently mandated

for use in all Department of Defense load planning. While more time consuming than AALPS, MPALPTS required the same or fewer aircraft than AALPS in all test scenarios. If implemented, MPALPTS has the potential to save AMC millions of dollars in airlift costs.

To my wife and four wonderful children

Acknowledgments

First and foremost, I would like to thank God for his ultimate wisdom and guidance throughout my time at AFIT. I would also like to thank my wife and children for putting up with my late nights and general lack of availability.

My advisors, Dr. August Roesener and Dr. Jim Moore, have been instrumental at guiding me through this very long process of coding and testing my algorithm as well as writing my thesis, and I am grateful for their expertise and advice. I would also like to thank Dr. Shane Hall for teaching me everything I know about the mighty tabu search and Dr. Kenneth Bauer for his teachings and encouragement as I learned and applied robust parameter design on a separate tabu search algorithm I authored.

Finally, I would like to thank the men and women of Air Mobility Command for their help in gathering data and giving me better a understanding the loading problem and its various constraints.

R. Larry Nance

Table of Contents

List of Tables

List of Figures

List of Equations

An Advanced Tabu Search Approach to Solving the Mixed Payload Airlift Loading Planning Problem

Chapter 1: Introduction

1.1 Global Airlift Power

Throughout military history, armies, large and small, have battled one another using dramatically different tactics and a wide range of weapons; however, one constant remains for every battle: the need for supplies. Sun-Tzu wrote, "So, armies cannot survive without supplies, cannot survive without provisions, cannot survive without stockpiled materials" (Huang, 1993). Without a steady flow of food, equipment and weapons, the military fighting machine could not function in ancient times; similar requirements hold for the conventional and unconventional wars currently being waged.

The United States Air Force (USAF) defines airlift as "the transportation of personnel and materiel through the air, which can be applied across the entire range of military operations to achieve or support objectives and can achieve tactical through strategic effects" (AFDD 1 2003). General Michael T. Mosely, a former USAF Chief of Staff, once wrote, "While other forms of American military power have some degree of inherent mobility, the scale of flexibility and responsiveness of the Air Force's air mobility forces is singular in the history of world conflict" (AFDD 2-6 2006). The ability to rapidly deploy vast numbers of both combat forces and their equipment to any point on the globe is a unique capability possessed by the United States (US); having this capability gives the US an unprecedented advantage in projecting power in the place and

at the time of its choosing. Unfortunately, airlift comes at a great price. In fiscal year 2008, the Air Force spent approximately $22,998 per flight hour to operate a C-5 and $12,911 per flight hour for a C-17 (Herbison 2008). In today's economically constrained environment, it is critical to operate these airlift missions in the most efficient way possible. Choosing the smallest number of aircraft required to move needed supplies and determining the exact placement of each piece of equipment on the aircraft is a very difficult problem to solve. As a result, this topic has been the focus of much research.

1.2 Current Airlift Process

The current airlift process begins with the United States Transportation Command (USTRANSCOM) tasking Air Mobility Command (AMC) with daily airlift requirements. Once tasked, planners in the Tanker Airlift Control Center (TACC) within AMC follow a continuous complex process to

> balance global requirements from its variety of government users against the availability and location of resources. This streamlined planning process focuses on a continuous, prioritized, frequently user-adjusted schedule rather than strategy creation and enemy analysis. For non-contingency situations, those taskings are rank ordered by priority and time received, then planned and executed (AFDD 2-6 2006).

Once the subset of priority missions is selected to be flown, planners begin the process of obtaining diplomatic airspace clearances, ascertaining airfield suitability and gaining host nation support. In many cases, load plans (which detail the exact positioning of cargo items in an aircraft) for USTRANSCOM tasked missions are created by the requesting agency using their own load planners. These load plans are validated by the appropriate validation cell within TACC. After validation, TACC tasks the missions to specific Air

Force Wings who, in turn, task the mission to specific airlift squadrons. The squadron assigns the required aircrew members to the mission.

The cargo requiring transportation will either be at the assigned wing's aerial port or at an aerial port at another location. Thus, the aircraft may either be loaded at its home station, or it may fly to a distant aerial port to be loaded. The aerial port is simply a distribution center for cargo and passengers. It contains personnel trained on preparing passengers and all types of cargo for flight. Prior to a mission's execution, specially trained personnel verify that TACC's load plan for the mission meets the various restrictions in place to ensure safety of flight, and they prepare the cargo for loading. On the day of the mission, the aerial port personnel generally load the aircraft prior to the aircrew arriving (if the cargo is at the aircraft's home station); the aircrew inspects the load and provides a final check to ensure the load is correctly positioned on the aircraft and meets all required restrictions. Once verified, the mission is flown and the cargo is delivered.

1.3 Cargo Types

Current military aircraft are equipped to carry a wide variety of cargo. One type is palletized cargo. Boxes and other suitably small items can be strategically loaded onto pallets, and a cargo net is secured on top of all the stacked items. The pallets are then loaded onto the aircraft such that specific weight and balance restrictions, discussed later, are satisfied. Each pallet, whose dimensions measure 88 inches by 108 inches and can be as tall as 96 inches, are packed such that its overall center of gravity falls in the center of the pallet. Pallets have predetermined locations available within the aircraft. Loading

occurs using a rail system on the floor of the aircraft; after pallets are loaded, the rail system secures them to prevent shifting during flight.

Military aircraft are also equipped to carry wheeled or tracked vehicles. These vehicles can range from tanks and trucks to helicopters and boats and are generally referred to as "rolling stock". To ensure the vehicles do not move during flight, they are secured using heavy duty chains connected to tiedown rings located on the aircraft floor.

1.4 Airlift Aircraft

The USAF has numerous aircraft capable of airlift activities. These include the KC-135, KC-10, C-130, C-17 and C-5. Generally, airlift capable aircraft can be classified into two categories: *inter*theater and *intra*theater. Intertheater airlift transports personnel and equipment from the Continental United States (CONUS) to a theater of operations or between two theaters (AFDD 2-6 2006). While the KC-135 and KC-10 can carry intertheater cargo and passengers, they have a much more limited cargo capacity than other intertheater airlifters and are more often utilized for their primary purpose of performing air refueling operations. Hence, the C-5 and C-17 are generally considered to be the primary intertheater airlifters (USAF 2008). Intratheater airlift operations move cargo and passengers inside a theater of operations and are generally supported by smaller tactical aircraft. The C-17, because of its ability to land on short unimproved runways, is often used as an intratheater airlifter, but the C-130 is the USAF's primary choice for intratheater airlift (USAF 2008). Because the focus of this research considers a large set of cargo requiring long-range transportation, only the intertheater C-17 and

C-5 aircraft are considered; however, the algorithms used are flexible enough to include other airlift capable aircraft.

1.5 Loading Constraints

In order to safely transport cargo, aircraft must be loaded to satisfy several constraints. For example, the Allowable Cabin Load (ACL), Center of Balance (CB) and chaining constraints are applicable to all airlift aircraft

The total gross weight of an aircraft includes the empty aircraft, the aircraft's fuel, and its cargo, passenger and crew load. The upper bound of gross weight is fixed by the structural and aerodynamic properties of the aircraft. There are two types of ACL. The first type (*planning* ACL) is the total weight of cargo and passengers that can be loaded on an aircraft under standard environmental and mission assumptions. Early in the planning process, personnel use the planning ACL as a non-strict upper bound; however, as weather forecasts and mission details are updated, this ACL constraint becomes binding. The second type (*maximum* ACL) is the maximum total weight an aircraft can carry and is primarily based upon the structural limitations of the aircraft floor. It is much larger than the planning ACL and rarely is a factor in loading aircraft. For this research, all references to ACL pertain to the planning ACL. Depending on the source document and the assumptions its author makes, the planning ACL for a C-5 and C-17 can vary. For this research, data from Air Mobility Command's airlift planning training manual is used; this document lists the C-5 and C-17 planning ACL as 150,000 pounds and 90,000 pounds, respectively (Air Mobility Command 2004).

In addition to the ACL, the CB constraint is critical to ensuring safe flight operations. An aircraft's center of gravity (CG) is affected by the amount of fuel, cargo and people in the aircraft. The cargo and passengers must be arranged in an aircraft such that the center or average of their weight falls within specific ranges inside the aircraft. The CB constraints are defined such that regardless of the amount of fuel required for the mission, the aircraft's CG will fall within acceptable ranges. Furthermore, there is also an optimal CB for a given cargo weight that minimizes fuel burn.

An additional constraint requires rolling stock to have approximately 24 inches of separation to allow sufficient space to secure the items to the floor of the cargo compartment. The actual space required is based upon the weight of the cargo item as well as the location of the item's tiedown rings; however, according to Senior Master Sergeant Tim Wakefield, a subject matter expert who is the chief C-5 loadmaster for Air Mobility Command, 24 inches is a very conservative estimate.

While the ACL, CB and chaining constraints can be modeled relatively easily in a mathematical program, aircraft specific constraints are much more difficult. These constraints generally encapsulate the strength of different parts of an aircraft's cargo floor. In the C-17 and C-5, the cargo compartment can viewed as having two columns: left and right. If a rolling stock item meets specific width and/or weight constraints, it can be loaded on one of the two columns resulting in the possibility of two items being adjacently loaded. If a rolling stock item is too heavy or too wide, it must be loaded in the center of the cargo compartment straddling the two columns. Pallets, on the other hand, are loaded into predefined pallet positions within the aircraft which are adjacently

located in each column of the cargo compartment. In their normal configuration, a C-17 has 18 pallet positions (9 in each column), while a C-5 has 36 positions (18 in each column). Each pallet position has associated maximum weight and height restrictions for any pallet placed there. To ease on-load and off-load, pallets can only be loaded aft of any rolling stock within a given column.

Cargo location within an aircraft is measured in inches by its flight station (FS). Every cargo aircraft has FS markings within the cargo compartment which measure the number of inches from the reference datum line (an imaginary point in front of the aircraft) to a specific point in the aircraft. For example, FS890 represents a distance of 890 inches from the reference datum line to a specific location in the cargo compartment, and FS990 represents a position that is exactly 100 inches aft of FS890. These measurements allow accurate CB calculations.

The FS markings also identify where cargo zones begin and end. Cargo zones define portions of the cargo compartment which have unique rolling stock weight capacities. The C-5 has seven cargo floor zones; the C-17 has four. Because rolling stock items may be adjacently loaded (one in each "column" of cargo compartment), there are also zone specific restrictions on the weight of the items' individual axles. If two axles on adjacently loaded rolling stock items are within an aircraft specific longitudinal distance of each other, their combined weigh must be below a particular zone specific value. In some aircraft, the acceptable adjacent axle weights are defined by a piecewise linear equation and are therefore extremely difficult to model in a pure mathematical programming problem.

Additionally, because the sides of cargo aircraft are curved inward, there are also specific height restrictions that must be met so the cargo does not touch the walls of the cargo compartment. For each inch taller than an aircraft specific limit, the cargo must be moved toward the center of the aircraft by a specific amount. For the purposes of this research, each cargo item is given a six inch buffer on each side to account for this restriction.

To simply produce a *feasible* loading of a large set of cargo items, load planners must ensure all of the ACL, CB and aircraft specific constraints are met; however, simply deriving a feasible solution is not an acceptable measure in today's high cost and restricted budget environment.

In military airlift, two main objectives exist. First and most importantly, military airlift must be effective; cargo must be delivered to the correct destination on schedule. The USAF does reasonably well at achieving airlift effectiveness. Efficiency is the second objective and is often overshadowed by the first objective. Data obtained from Air Mobility Command's Analysis division, which included approximately 1,480 C-5 and 11,280 C-17 flights flown from 1 Jan 08 to 30 Sep 08, indicated the C-5 and C-17 carried an average cargo load of 51,033.35 and 35,595.55 pounds, respectively (Anderson 2008). These averages are approximately one-third of their planning ACL. Histograms of each airframe are shown in Figures 1 and 2 below.

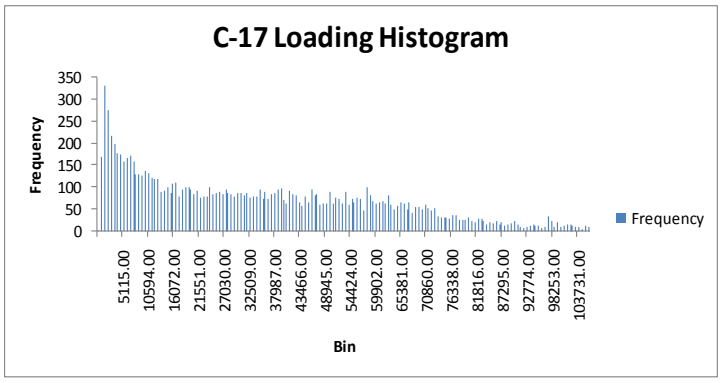


Figure 1. C-17 Loading Histogram

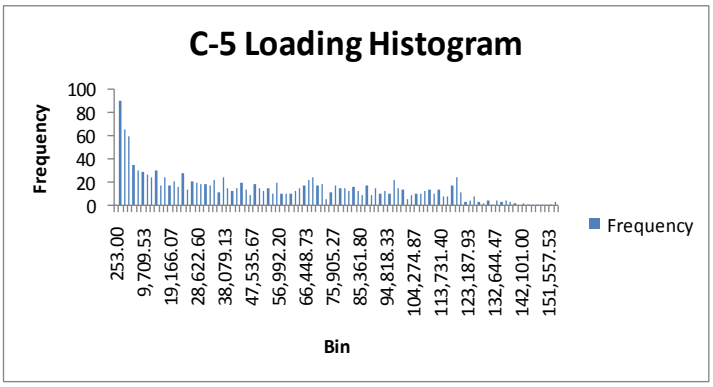


Figure 2. C-5 Loading Histogram

It is important to note that achieving an average loading at or near 100% of an aircraft's ACL is probably not possible. For example, the data presented includes unavoidable pre-positioning and de-positioning flights where the aircraft are flying empty to go pick up cargo or returning home after delivering cargo to its destination. Additionally, the C-17 often reaches space limitations before its entire ACL is used making efficiency calculations solely based on ACL less valid. Despite these caveats, the data does indicate a need for efficiency improvements. An algorithm which quickly

produces high quality solutions using the smallest number of aircraft while minimizing fuel burn would enhance the USAF's ability to effectively *and* efficiently utilize its airlift prowess. Unfortunately, with billions (or more) possible combinations, finding high quality solutions is a challenge.

1.6 Current Load Planning Software

Because of the complexity of this problem, the Department of Defense (DoD), currently uses a commercial software package called Automated Air Load Planning System (AALPS) to automate this process. AALPS uses preplanned and actual data for estimating airlift requirements for a given set of cargo (including pallets and rolling stock) and creates feasible load plans for each aircraft's load. It has models for nearly every commercial and military airlift aircraft and considers all ACL, CB and aircraft specific constraints. It also has a very large database of military equipment making data input relatively simple (USTRANSCOM 2007). Unfortunately, it has been shown that, for palletized cargo, AALPS' greedy heuristic generates inferior solutions when compared to a tabu search heuristic (Roesener 2006).

1.7 Research Objectives and Assumptions

The primary goal of this research is to develop and validate a tabu search-based algorithm which, given a set of cargo items and a set of available C-5 and/or C-17 aircraft, creates feasible load plans for a minimal number of aircraft such that all of the required rolling stock and pallets are loaded. This problem is called the Mixed Payload Airlift Load Planning (MPALP) problem. To be valuable, this MPALP Tabu Search (MPALPTS) algorithm must provide a better solution than AALPS. Specifically, given a

set of cargo, this heuristic must produce a feasible loading of all cargo which requires no more (and preferably fewer) aircraft than AALPS. This feasible solution must account for aircraft ACL, CB, cargo securing requirements and all aircraft specific loading constraints.

Given a problem of this scope, there are several assumptions which must be made to clearly define the problem:

1. Hazardous Cargo: The cargo to be loaded does not have any hazardous cargo-related loading restrictions.
2. Outsized Cargo: Extremely outsized cargo such as helicopters or large boats often have specific loading instructions outlined in Air Force Technical Orders. The research assumes the set of available cargo does not have special loading requirements (such as a need to rotate an item in the cargo compartment) and that each item will physically fit in the cargo compartment; there are a very limited number of unique ways to load outsized cargo, and the problem therefore becomes trivial.
3. Shoring: Some cargo requires shoring which is small planks of plywood stacked on top of each other. Shoring is placed under the treads or wheels of some rolling stock items to help distribute its load over a larger area and protect the cargo floor from damage. The research assumes aerial port or loadmaster personnel properly place shoring when required.
4. All cargo has the same destination.
5. The set of available aircraft only includes C-5 and/or C-17 aircraft.
6. In some cargo aircraft, loadmasters prefer to back rolling stock items into the aircraft to facilitate faster offloads. In this algorithm, rolling stock items all face forward in the cargo compartment.

The following chapters demonstrate a new tabu search heuristic that consistently and quickly produces feasible load plans using fewer aircraft than AALPS. The heuristic

also outputs the placement of each item of rolling stock in each aircraft thereby giving load planners a complete picture of the best way to load the cargo.

Chapter 2: Literature Review

2.1 Heuristics Motivation

2.1.1 Classes of Combinatorial Optimization Problems

A combinatorial optimization problem (COP) is said to be solvable in polynomial time if the time to reach the optimal solution is bounded above by some polynomial function of the size of the COP instance. Informally, the decision problem of a COP answers the question "does there exist a primal solution value as good or better than [some value] *k,"* and solving the decision problem a polynomial number of times will optimally solve the associated COP (Woosley 1998). Hence, a particular COP's decision problem (and therefore the COP itself) can be classified into several categories. First, the non-deterministic polynomial class (*NP*) defines the class of decision problems for which a "Yes" answer can be obtained and proven with a polynomial proof (Woosley 1998). A subset of NP is the class of "easy" decision problems (*P)* for which there exists a polynomial algorithm (Woosley 1998). All COPs whose decision problem falls in *P* can be solved to optimality in polynomial time. Finally, the class NP-complete (*NPC)* can be thought of the "hard" decision problems and is known to be non-empty (Woosley 1998). To prove a decision problem is a member of *NPC*, a problem already known to be in *NPC* must be shown to be polynomially reducible to the problem of interest. Thus, when a COP is said to be NP-hard, its associated decision problem is a member of *NPC* which implies there exists no known polynomial algorithm which solves the COP to optimality.

2.1.2 MPALPTS Complexity

The MPALP has received a moderate amount of academic attention in the past 15 years. In a broad sense, research has focused on two basic approaches both of which are discussed in detail later. The first approach is based on a multidimensional knapsack problem; it assumes each item of cargo has an associated priority, and a limited number of aircraft are available to transport a subset of the cargo. Hence, the knapsack solution loads as much cargo as possible on the available aircraft leaving some cargo unloaded. The second approach uses bin packing techniques to load all cargo items on a minimal set of aircraft. Because both underlying knapsack and bin packing problems have been shown to be NP-hard (Garey and Johnson 1979), and the MPALP adds significant complexity to the problem through the additional constraints (CB, ACL, aircraft specific, etc.), the MPALP is most likely also NP-hard. No formal proof of this claim is presented.

2.1.3 "Solving" an NP-hard problem

By definition, optimally solving instances of NP-hard COPs cannot be done in polynomial time, so it is often useful to trade a guaranteed optimal solution for a high quality solution in polynomial time. Some authors argue that there are very few real-world instances of COPs which modern computing power cannot solve in reasonable time (Goulimis 2007); however, the definition of reasonable time often depends on the situation. The greatest obstacle to optimality for the MPALP is the practical need for speedy solutions. It is not uncommon for last minute changes to be made to the load plans due to unforeseen circumstances. These circumstances can range from a faulty cargo item that is unable to be loaded onto an aircraft (leaking fluids, engine troubles,

etc.) to a last-minute high priority cargo item that must be included in the load. This drives the need to generate a new load plan in minimal time (minutes not days) and motivates the use of a heuristic-based solution.

2.2 Heuristics

Pearl describes a heuristic, in the most basic sense, as rule or collection of rules that guides one's actions with the intent of finding a solution to a problem (Pearl 1984). Unfortunately, general heuristics do not necessarily guarantee an optimal or even feasible solution to a problem; however, two broad categories of heuristics do guarantee at least a feasible solution: approximation algorithms and searched-based algorithms.

Approximation algorithms guarantee both polynomial time execution and feasibility, and most can be shown to have a worst-case objective function value bound in terms of the optimal solution. They follow a series of defined steps (an algorithm) using a rule of thumb to exploit some structure in the problem to find high quality solutions. While there is no guarantee of optimality, solutions from approximation algorithms can be quite close to the optimal solution.

On the other hand, search-based heuristics are akin to finding a minimum or maximum function value to a mathematical equation using derivative and gradient information. These heuristics can find the global optimal solution or a local optimal solution. Given an initial starting point which may or may not be feasible, the generic local search heuristic attempts to find a feasible solution with an improved objective function value (Aarts and Lenstra 1997). Like approximation algorithms, the final solution is guaranteed to be feasible but not guaranteed to be optimal.

2.3 Tabu Search

In literature, the search-based heuristic called tabu search is often used to find high quality solutions to MPALP-related instances. The tabu search is an iterative improvement algorithm which chooses the next solution as the best feasible solution that neighbors the current solution in some manner. This next solution is not required to be an improving search direction nor is it even required to be feasible, thereby allowing for the ability to circumnavigate local optima. The feasible neighbors of a solution are restricted by a dynamically updated tabu list which prevents returning to recently explored areas; the notion of recency in this context is formally defined as the tabu tenure and can be a fixed or adaptively varying value. The challenge with a tabu search and many other search-based heuristics is the need to tailor the parameters of the search such that high quality solutions are found for any problem instance (Aarts and Lenstra 1997). Therefore, it is important to "fine-tune" the algorithm through robust parameter design techniques. Glover presents a more detailed explanation of tabu search and the applications to which it has been applied (Glover 1989).

2.4 Knapsack MPALP Instances

The single dimensional knapsack problem can be thought of as a set of items with associated profits and weights. A subset of these items must be placed in a knapsack with a limited weight capacity in such a manner to maximize the overall profit. The integer program is formulated as

$$\max \sum_{i=1}^{n} c_i x_i$$
$$s.t.$$
$$\sum_{i=1}^{n} w_i x_i \le b$$
$$x_i \in \{0,1\}, c_i, w_i, b \in Z^+ \quad i = 1,2...,n$$

Figure 3. Knapsack Integer Program

where $x_i = \begin{cases} 1 \text{ if item i is placed in the knapsack} \\ 0 \text{ otherwise} \end{cases}$, c_i is the i^{th} item's profit, w_i is the i^{th} item's weight and b is the weight capacity of the knapsack. In multidimensional knapsack problems, additional constraints, such as the actual size or volume of each item, are included in the formulation. Geometric knapsack problems capture the shape of each item and the effect the shape has on the ability of items to fit into the knapsack. In relation to the MPALP, each aircraft can be represented as a knapsack, and each cargo item can be represented in the set of items which can be placed into the aircraft. By the problem definition, more cargo are items available than the set of aircraft can feasibly carry.

2.4.1 Tabu Search Knapsack MPALP

In his research, Chocolaad used a tabu search based heuristic to find quality solutions to the geometric knapsack MPALP using a single aircraft and a set of cargo items (Chocolaad 1998). Cargo items were prioritized based on their weights; a knapsack heuristic selected cargo in each iteration while a packing heuristic determined feasibility.

The knapsack heuristic was based on a critical event tabu search which alternates between constructive and destructive phases. The constructive phase adds items to the

aircraft while the destructive phase removes them. The critical event terminates the heuristic at the last solution obtained either at the step before a constructive phase search enters an infeasible region or when the first feasible solution is reached during the destructive phase. As the knapsack heuristic searches for items to place in the aircraft, the packing heuristic employs a simple tabu thresholding local search method to check feasibility requirements. This approach allows non-improving moves to avoid becoming trapped at a local optimum while ensuring feasibility with respect to CB and aircraft specific weight constraints (Chocolaad 1998).

While Chocolaad's approach was effective in finding quality solutions, the heuristic's output only indicated which cargo items were assigned to the aircraft and did not include where the items were placed inside the aircraft. Further, it was limited to loading one aircraft at a time rather than loading multiple aircraft simultaneously. Romaine later expanded Chocolaad's work by adding the possibility of loading multiple aircraft simultaneously and removing the implied homogenous aircraft restriction, but the exact position of each cargo item was still not defined in the heuristic's output (Romaine 1999).

2.4.1.1 Multiple Choice Multiple Dimension Knapsack Problem (MMKP) Heuristic

Hiremath and Hill define the MMKP as a knapsack problem consisting of multiple classes of items and multiple knapsacks; the objective is to select exactly one item from each class while maintaining the knapsack constraints (Hiremath and Hill 2007). The approach generated an initial solution using a gradient based heuristic called NG V3 and then refined this solution using a local search heuristic. After constructing

the initial solution, they utilized two different neighborhood generating functions to improve the solution. First, they considered exchanging every item in a given class with the current solution's item. They picked the best feasible solution which yielded an improved objective function value and repeated this process until either a maximum number of iterations were reached or every class was fully explored. In the second function, Hiremath and Hill performed the same basic search; however, when the search resulted in a series of non-improving objective function values (i.e. a cycle or a local optima), they abandoned the search in favor of their DoubleSwap neighborhood. The DoubleSwap considered two-tuples of classes and searched through both classes, selecting an item from each class per iteration. If swapping the two items with two items of the same classes from the current solution yielded a better objective function value, the DoubleSwap selected that solution and continued searching in other two-tuples of classes. This search repeated until all two-tuples had been explored or until a maximum number of iterations had been reached (Hiremath and Hill 2007).

Their results showed, in general, the second approach created better solutions to an established set of problem instances and often outperformed many of the leading heuristic approaches (Hiremath and Hill 2007). While this algorithm does not consider the MPALP, the MMKP structure is similar to the basic MPALP; thus their choice of neighborhoods may have merit in generating better MPALP solutions.

While knapsack MPALP instances are adept at loading the high-priority portion of cargo on a given set of aircraft, many airlift taskings require that all of the cargo be loaded. For these types of problems, bin packing formulations are preferred.

2.5 Bin Packing Problems

Unlike the knapsack approach, a bin packing problem attempts to load the entire set of items into a minimal number of bins. In relation to the MPALP, all of the cargo items must be loaded in a minimal number of aircraft. Like knapsack formulations, multidimensional and geometric versions of the bin packing problem have also been extensively studied. For a more detailed overview of bin packing problems and some associated approximation algorithms, see (Vazirani 2003) or (Lodi, Martello and Vigo 2002).

2.5.1 Bin Packing Instances

This research's primary goal is to develop a tabu search-based heuristic to find premium solutions to bin packing MPALP instances. While much research has been done in this area, most do so by assuming away difficult aircraft specific constraints or implementing less effective heuristic-based algorithms.

2.5.1.1 Goal Integer Programming Method

In 1989, Kevin Ng examined the bin-packing MPALP for Canada's C-130 aircraft (Ng 1992). His goal was to move a set of cargo items including pallets and rolling stock on a minimum number of C-130 aircraft. To solve this problem, he used a pre-generated set of 38 "standard" cargo loadings which were subsets of the overall items to be airlifted. These standard cargo loadings were certified by Canadian C-130 loadmasters. Using goal programming, Ng created three priorities which included: all items had to be airlifted, a minimum number of aircraft was used and the weight of excess capability was

maximized. This technique gave him the ability to solve the problem to optimality using branch-and-bound and reduced the required number of aircraft from the manual solution of 121 to 110 saving the Canadian Air Force $1.21 million (Ng 1992).

While this approach does have merit, it requires a large set of pre-generated load plans covering the entire set of cargo to be transported. As Ng pointed out, "there are severe limitations to using standard loads. Many items do not have standard dimensions…The model, in its current form does not have the flexibility to modify load plans" (Ng 1992). While creating template loads for all the vast number and types of cargo items the USAF airlifts is impractical, AALPS uses template loadings for commonly airlifted items with the goal of producing higher quality solutions.

2.5.1.2 Two-Dimensional Orthogonal Packing

Unlike Ng, J.M. Harwig et al. developed an adaptive tabu search algorithm for two-dimensional orthogonal bin packing (Harwig, Barnes and Moore 2006). While their problem formulation was not specific to aircraft operations and therefore does not account for CB or floor weight restrictions, their choice of objective function and searching methods merit study.

Harwig et al. created a "fine-grained" objective function which effectively evaluated competing moves within the solution space (Harwig, Barnes and Moore 2006). The objective function favored the three search moves which were likely to decrease the overall number of bins. First, they designated an excess bin whose function was to hold items that were not otherwise packed. When the excess bin was emptied, it was discarded and a new excess bin was designated. Thus, moves which removed items from

the excess bin and placed them in other bins were highly favored. Next, an *intra*-bin move was attempted to produce a more compactly packed bin. Finally, they incorporated an *inter*-bin move which created a large "dead space" within the losing bin; the goal of this move was to allow the relocation of an item from the excess bin (Harwig, Barnes and Moore 2006).

Items in the excess bin were also penalized with a unique two-dimensional potential energy-based function (Harwig, Barnes and Moore 2006). This penalty function was defined in terms of the height, length and weight of the items within the bin. Specifically, smaller items were penalized less than larger items because they are much easier to pack into the other bins.

In addition to the previously described neighborhoods, Harwig et al. employed ejection chains. When an insert move places a new item in a bin, the result may be the ejection of another item from the bin. An ejected item is then feasibly placed inside the excess bin. Finally, they use inter and intra-bin swaps as a means to search through large portions of the solution space.

When used on a common test set of 500 problems, the algorithm of Harwig et al. improved on the previously best solutions by an average of 25%. Unfortunately, this algorithm cannot be directly applied to aircraft loading because it would only guarantee feasibility with respect to the space inside an aircraft; CG and zone constraints would not necessarily be satisfied. While their bin packing algorithm would not necessarily produce feasible results in loading aircraft, their methods may be applicable to this research endeavor.

2.5.1.3 Barrier Based Tabu Search

Heidelberg et al. developed a heuristic-based bin-backing algorithm which, "often matches the capability of expert loadmasters usually requiring no adjustment for better packing efficiency" (Heidelberg, Parnell and Ames IV 1998). They first show classical bin-packing heuristics such as level-based algorithms are inadequate at loading aircraft because they tend to pack the heaviest items towards the left-aft portion of the bin, which, in an aircraft, would create CB violations. Instead, Heidelberg et al. proposed a packing barrier approach which uses a barrier (Figure 4) which can be bent at 90° angles at a maximum of four places (Heidelberg, Parnell and Ames IV 1998).

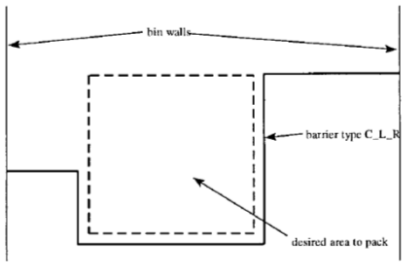


Figure 4. Heidelberg et al. candidate bin selection

The packing barrier approach can be described in the following manner. First, the algorithm determines where, relative to the current barrier configuration, an item should be placed. Next, it chooses the best candidate item to be packed in that section; after placing the item, the algorithm determines the best shape of the new barrier. As the aircraft fills with cargo, the probability that subsequent items will exceed the space or weight capacity of the aircraft increases. When no additional items can be loaded on the aircraft, the barrier's shape is changed in an attempt to accommodate a larger item. Once all barrier combinations have been tried, the aircraft is deemed full.

Heidelberg et al. tested this algorithm on two sets of random sized cargo items constructed to represent typically sized military cargo and compared its performance to the CLS (Constrained Local Search) and BFLD (Best Fit Level Decreasing) algorithms (K. Heidelberg 1995) (Gehring 1990). The first test set consisted of a random number of items. Each item's size was also randomly generated. The three methods were compared using 20 different instances from the first test set. The second set was designed to be more representative of actual aircraft loads. It included a random number of a smaller set of randomly sized items each of which was duplicated between five and ten times. The three methods were also compared using 20 instances from the second test set.

In the 20 trials of cargo loads from the first test set, their algorithm outperformed CLS and BFLD in every instance. CLS outperformed their algorithm once and BFLD tied it twice in the 20 trials of cargo loads from the second test set (Heidelberg, Parnell and Ames IV 1998). In both cases, their algorithm took longer to find a solution than the other two; however, the worst-case running time in any of their test replications was under two minutes.

Unfortunately, Heidelberg et al. did not elaborate on the methods they used to handle aircraft specific constraints such as ACL, zone restrictions or even the space required to chain cargo items to the floor; however, their heuristic must account for these constraints because AALPS uses it to produce feasible load plans (Heidelberg, Parnell and Ames IV 1998). Despite its operational success, Roesener showed AALPS to be less effective than a tabu search based approach with respect to loading pallets (Roesener 2006).

2.5.1.4 MPALP Specific Tabu Search

Roesener developed a tabu search based heuristic to solve a portion of the MPALP which accounts for CB, ACL and aircraft specific constraints (Roesener 2006). This research is strongly related to his work; however, he focused on loading pallets while this research loads both pallets and rolling stock. There are four areas of Roesener's research that are of interest: the initial solution, tabu list construction, objective function calculations and neighborhood search functions.

Roesener began by determining the upper and lower bounds on the number of aircraft required to load the pallets (Roesener 2006). The upper bound came from an AALPS solution to the same problem instance which was known to be feasible. For the lower bound, he used the following equation:

$$\max\left(\left\lceil \frac{\text{total number of pallets}}{\text{number of pallet positions in aicraft}} \right\rceil, \left\lceil \frac{\text{total pallet weight}}{\text{aircraft ACL}} \right\rceil\right)$$

Equation 1. Roesener's Lower Bound Calculation

This equation defines a relaxation of the problem by possibly allowing CB violations resulting in a lower bound to the number of required aircraft.

Using the upper bound as the initial set of available aircraft, Roesener developed an algorithm which constructs an initial solution by simultaneously maximizing both volume and ACL. First, the pallets are placed into the "Big Bin" (which is similar to Harwig's excess bin) and then the pallets are sorted by weight from heaviest to lightest (Roesener 2006), (Harwig, Barnes and Moore 2006). The sorted pallets are then divided into equal sized groups. The available and empty aircraft with the largest ratio of ACL to

open pallet positions is selected for loading. Pallets are iteratively loaded into the selected aircraft by placing the heaviest pallet in each group on the aircraft until either the aircraft has no open pallet positions or has reached its planning ACL. The next step is to refine this aircraft's loading. If all pallet positions are used but there is excess ACL remaining, the lightest pallet on the aircraft is replaced with the heaviest pallet in the Big Bin that does not violate the aircraft's ACL. On the other hand, if there are open pallet positions but the planning ACL has been reached, the heaviest pallet on the aircraft is removed, and as many light pallets as possible are placed onto the aircraft. This cycle continues until either all the aircraft have been fully loaded as determined by the algorithm or the Big Bin is empty.

It is important to note this initial solution is not guaranteed to be feasible with respect to CB or aircraft specific constraints, and the solution may not even load all of the cargo on the aircraft. The algorithm's tabu search is created such that it will seek feasibility for these constrains as it tries to decrease the number of required aircraft.

For the tabu tenure, Roesener selected a dynamic value which decreases with improving moves and increases with worsening moves. The initial value for the tabu tenure was calculated based upon the number of pallets available for loading (Roesener 2006). This effectively intensified searches within areas of improvement and diversified in areas of poor quality solutions (Roesener 2006).

The quality of any particular solution was determined using a five part additive objective function which penalizes infeasibility and undesired search locations. The overall goal was to minimize the resulting objective function value. Each part of the

objective function has a single or multiple weights, which are set to drive the search toward desirable and feasible solutions. First, Roesener penalized unloaded pallets (i.e. any pallets remaining in the Big Bin) using the following equation:

$$\lambda_1 \sum_{i=1}^{N} B_i,\ B_i \in \{0,1\} \forall i = 1,...,N$$

Equation 2. Roesener's Unloaded Pallet Penalty Function

This equation simply assigns a penalty of λ_1 to each pallet which remains in the Big Bin. In this equation, N equates to the total number of pallets and B_i is a binary variable which indicates whether a pallet has been loaded (Roesener 2006).

Next, Roesener penalized each aircraft utilized in the solution. This drives the tabu search into areas where fewer aircraft were required. This was accomplished with:

$$\sum_{j=1}^{M} C_j A_j,\ A_j \in \{0,1\}\ \forall\ j = 1,2,...,M, C_j \in R$$

Equation 3. Roesener's Aircraft Cost

In Equation 3, M represents the total number of aircraft available, and A_j is a binary variable indicating whether the j^{th} aircraft is used in the solution. This equation gives load planners extra flexibility by allowing individual cost coefficients to be assigned to individual aircraft or aircraft types. It also prevents the loading of unnecessary aircraft by ensuring the volume of each loaded aircraft is utilized to the maximum extent possible (Roesener 2006).

One goal of Roesener's objective function is to maximize the utility of each aircraft's ACL. Hence, there is a penalty assigned to aircraft whose ACL is not 100

percent utilized. Roesener viewed ACL as a non-strict upper bound and allowed aircraft to be overloaded by up to 2.5%. Thus, the ACL penalty function has two parts: under-loading and overloading.

$$\lambda_2 \sum_{j=1}^{M} 100 - \%WF^{\ 2} A_j X_j,\ A_j \in \{0,1\}\ \forall\ \text{j=1,2,...,M},\ X_j \in \{0,1\}\ \forall\ \text{j=1,2,...,M}$$

$$\lambda_2 \lambda_3 \sum_{j=1}^{M} 100 - \%WF^{\ 2} A_j (1 - X_j),\ A_j \in \{0,1\}\ \forall\ \text{j=1,2,...,M},\ X_j \in \{0,1\}\ \forall\ \text{j=1,2,...,M}$$

Equation 4. Roesener's Under-loading and Overloading Penalty Functions

As before, M represents the number of aircraft and A_j indicates whether aircraft j is used in the solution, while $X_j = 1$ if the j^{th} aircraft has unused ACL and equals zero otherwise. Notice that both functions assign a squared penalty for deviating from the ACL and the overloading penalty is penalized more heavily than the under-loading penalty when $\lambda_3 > 1$ (Roesener 2006). This ensures solutions with underloaded aircraft are significantly better than alternate solutions while still allowing the possibility of overloading.

Roesener also included penalty functions for lateral and longitudinal CB requirements. As mentioned in section 1.5, longitudinal CB constraints are critical to safe aircraft operations and include a desired position within the upper and lower bounds where fuel consumption is minimized. Lateral CB constraints (i.e. balancing the load relative to a centerline drawn down the length of the aircraft) are not normally considered in the load planning process because the aircraft pilot can compensate for nearly any lateral imbalance; however, Roesener included this calculation as an added benefit of his method. These portions of the objective function are defined in Equations 5 and 6:

$$\lambda_4 \sum_{j=1}^{M} \left(Lat\_CB_j \right)^2 A_j,\ A_j \in \{0,1\}\ \forall j = 1,2,...,M$$

Equation 5. Roesener's Lateral CB Penalty Function

$$\lambda_5 \sum_{j=1}^{M} (\text{Target\_Long\_CB}_j\text{-Long\_CB}_j)^2 A_j Y_j$$
$$+\lambda_5 \lambda_6 \sum_{j=1}^{M} (\text{Target\_Long\_CB}_j\text{-Long\_CB}_j)^2 A_j (1 - Y_j)$$
$$A_j \in \{0,1\}\ \forall j = 1,2,...,M,\ Y_j \in \{0,1\}\ \forall j = 1,2,...,M$$

Equation 6. Roesener's Longitudinal CB Penalty Function

In Equation 5, the desired lateral CB is zero, so any deviation from zero is penalized. For Equation 6, Y_j = 1 when the CB is within acceptable limits, and 0 when it is not. Thus, the equation penalizes an acceptable but less than desired CB much less than an infeasible CB; higher quality solutions are feasible and close to the target CB (Roesener 2006).

After the initial solution and its associated objective function value is calculated, Roesener's main tabu search begins. It combines four strategically chosen neighborhood functions which are similar to Chocolaad's and Harwig's. The first function, Big Bin to Aircraft Insert, is performed only if the initial solution leaves any pallets in the Big Bin. It selects the heaviest pallet in the Big Bin and places it into an empty pallet position on the aircraft which has the greatest available ACL. If all of the aircraft have maximized ACLs, it puts the pallet on the aircraft which would create the smallest ACL violation. This process is repeated until the Big Bin is empty.

While traversing the solution space, if an aircraft becomes trivially loaded, which Roesener defines as utilizing less than 25% of the ACL, or if the search becomes

stagnated, a diversification neighborhood is invoked called Unload Entire Aircraft. This neighborhood empties an entire aircraft by using the same logic as the previously defined Big Bin to Aircraft Insert move; unloading ceases if every remaining aircraft has no vacant pallet positions.

If the current solution is infeasible with respect to CB or aircraft specific constraints, the Intra-Aircraft Insert/Swap neighborhood function is chosen. In this neighborhood, the algorithm calculates the objective function value of every possible combination of swapping two pallets or moving a pallet into a currently empty position within a single aircraft. The best non-tabu permutation of the current solution is chosen as the new solution; iterations of this neighborhood typically produce a feasible loading.

The final neighborhood is the Inter-Aircraft Insert/Swap Neighborhood. This neighborhood is used as the primary means of traversing the solution space. In this move, the algorithm calculates objective function values for every possible swap of two pallets or a pallet and an empty position between two non-empty aircraft. The algorithm picks the best non-tabu solution as the next move.

To test his tabu search, Roesener executed twelve scenarios each of which varied levels within the number of pallets, type(s) of aircraft, or the distribution of pallet weights. In nearly every trial, his algorithm decreased the number of required aircraft from the best AALPS solution; however, his search techniques required more time to generate the solution than AALPS. Unlike AALPS, Roesener's algorithm returns solutions that were deemed feasible, trivially infeasible and marginally infeasible. As mentioned before, ACL can be viewed as a loose upper bound, so his trivially and

marginally infeasible solutions allow the ACL to be exceeded by a maximum of 1.5% and 2.5%, respectively. This gives the planner extra flexibility to decide if slightly exceeding the ACL is worth the potential of using fewer aircraft (Roesener 2006).

2.5.2 Summary

This chapter presented research which has been previously conducted in the area of tabu search in general and aircraft loading in particular. The knowledge of previously conducted research enables this thesis to focus on new methods of finding MPALP solutions while preventing duplication of efforts.

Chapter 3: Methodology

3.1 MPALP Tabu Search

Much of the MPALPTS presented here is based off of Roesener's work; however, the unique complexities of feasibly loading rolling stock requires additional objective function costs and an efficient method of packing the cargo to meet CB and zone constraints (Roesener 2006).

3.2 Decision Variable Definition

The primary decision variable within MPALPTS is a four-dimensional matrix defined as:

$$x_{ijk} = [\text{FSFwd}, \text{FSAft}, \text{CargoID}, \text{Centered}]$$
$$\text{where } i = 1,\ldots,A \ , j = 1,2 \ , k = 1,\ldots,c_{ij}$$
$$\text{and } \sum_{i=1}^{A}\sum_{j=1}^{2} c_{ij} = C, \quad \text{FSFwd}, \text{FSAft}, \text{CargoID} \in Z^{+}$$

Equation 7. Decision Variable

In Equation 7, *A* represents the number of aircraft used in the current solution and is indexed by *i*; *j* represents the number of columns in the aircraft where one represents the left column and two represents the right column; *k* represents a specific cargo item loaded on the i^{th} aircraft in column *j*. The total number of cargo items, *C*, can be determined by summing all the cargo items in each column of each aircraft. "FSFwd" and "FSAft" represent the FS where the front and back of the cargo item is loaded. "CargoID" represents the identification of the cargo item itself, and "Centered" is binary

such that a value of 1 indicates the specific item is centerline loaded. For example, $x_{1,2,4} = [512, 612, 9, 0]$ indicates that cargo item nine is the fourth item loaded on the right column of aircraft one from FS512 to FS612 and is not centerline loaded. This variable is used to represents the current solution within the MPALPTS.

3.3 MPALPTS Input Tables

The MPALPTS uses Microsoft Excel spreadsheets to represent all characteristics of the available aircraft and cargo and allows the user to import this data directly. Where possible, the actual values within each table are shown; however, due to DoD security policies, some aircraft-specific data is removed.

3.3.1 Aircraft Representation

The aircraft available to be loaded is represented as a A x 19 matrix with the following information:

Table 1. Available Aircraft Table

Acft Type (1)	ACL (2)	FS Min (3)	FS Max (4)	Col Width (5)	# Cols (6)	# Zones (7)	Tail # (8)	%Sp (9)	%Wt (10)	Useage Fee (11)	# Lft (12)	# Rt (13)	# Ramps (14)	Aft Ramp FS Min (15)	Aft Ramp FS Max (16)	Fwd Ramp FS Min (17)	Fwd Ramp FS Max (18)	# Pallet Pos (19)
1	150000	395	2131	114	2	7	9007	0	0	1000	0	0	2	1971	2131	395	517	18
2	90000	390	1403	106.5	2	4	1468	0	0	1000	0	0	1	1165	1403	0	0	9

This table defines the basic characteristics of each aircraft as well as provides a location for storing general information about the cargo loaded on the aircraft. An aircraft type (1) of one indicates a C-5 aircraft, and a two indicates a C-17. The aircraft's ACL (2), cargo compartment dimensions (3, 4), number and width of any columns (5, 6), the

number of zones (7), and the number and locations of any ramps (14 to 19) are all defined in this table.

3.3.2 Cargo Representation

Cargo representation is accomplished with a similar $C \times 20$ matrix whose columns are shown in Table 2:

Table 2. Cargo Representation

L	W	H	# Axel	Axle Location						Axle Weight						Total Wt	CB	ID	Type	Descrip-tion
				1	2	3	4	5	6	1	2	3	4	5	6					
190	106	106	0	0	0	0	0	0	0	0	0	0	0	0	0	23500	92	25	1	Vehicle
294	125	142	4	63	111	167	215	0	0	9480	9500	11280	10900	0	0	41160	143	26	1	Truck
88	108	70	0	0	0	0	0	0	0	0	0	0	0	0	0	2500	0	10	2	Pallet

The first four columns describe the length, width, height (in inches) and the number of axles of the cargo item. The next six columns represent the measurement from the front of the cargo item to each of up to six axle locations (at the time of this research, very few rolling stock items had more than six axles), and the subsequent six columns give the weights of all the axles. Tracked vehicles and pallets have zero axles and zero axle weights. The total weight is placed in the 17th column. The CB represents the center of balance of the cargo item (rolling stock with axles) measured in inches from the front of the cargo item and is computed by:

$$CB = \frac{\sum_{j=1}^{6} \text{Axle Location}_j \quad \text{AxleWeight}_j}{\text{Item's Total Weight}}$$

Equation 8. CB Calculation for Cargo with Axles

For tracked vehicles, the CB is predetermined and specified for the loadmaster because there are no traditional axles present for manual calculation. A pallet's CB is assumed to be in the geometric center of the pallet position in which it is placed. The last columns of Table 2 are the "Cargo ID" (which is a unique index for every cargo item), "Type" (which is set to one for rolling stock and two for pallets), and "Description" (which is a text field specifying the name of the cargo).

3.3.3 Zone Representation

Aircraft cargo zones define the weight restrictions of rolling stock items within specific zones of the cargo compartment. Table 3 shows the header information contained within the MPALPTS zone table and defines C-5 and C-17 zone constraints.

Table 3. Zone Representation

Cargo Zones C-5											
Zone	# Sub-Zone	FS Lower (1)	FS Upper (2)	Max Wt Single Axle (3)	Axle Wt Min (4)	Axle Wt Max (5)	Coef (6)	Intercept (7)	In Length (8)	Center-line (9)	Max Total Wt (10)

The "Zone" column represents the zone and "Subzones" represents the number of subzones that are within that zone. Columns (1) and (2) represent the fore and aft FS boundaries of the zone while "Max Weight Single Axle" represents the allowable maximum weight for a single axle within the zone. The "Centerline" column defines the

weight of an axle within the zone which, if exceeded, requires the cargo item to be centerline loaded while the "Max Tot Wt" is the maximum total allowable weight for all the cargo within that zone. The weight of adjacently loaded axles within a specified distance of each other ("In Length" column) can be restricted by the piecewise linear equations such as the one shown in Figure 5.

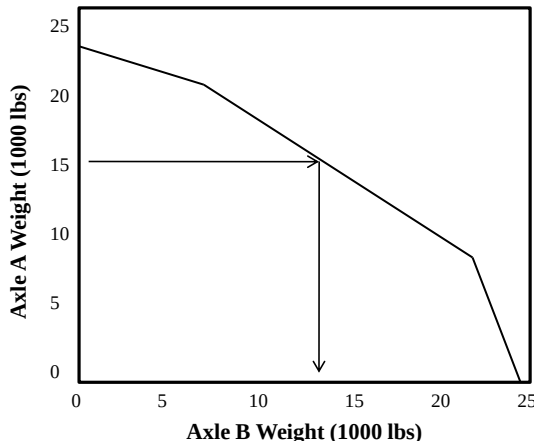


Figure 5. Representation of Adjacently Loaded Axle Constraints

Figure 5 represents the weight restrictions placed on two adjacent axles in a particular zone that are within the specified distance of each other. One enters the chart on the left side at the axle weight of the heaviest axle and travels horizontally until reaching the bold line. The maximum amount of weight allowed for the lighter axle is read on the bottom scale. For example, if the heaviest axle weighs 15,000 pounds, the maximum weight of the adjacent axle is 13,000 pounds. In order to model this piecewise linear relationship, the equation of each line segment (the slope and the intercept) was calculated and called a subzone. Each subzone is defined by the range of weights each

line segment represents which is recorded in columns (4) and (5) of Table 3. The allowable weight of an ajacently loaded axle can be determined from Equation 9.

$$\text{Allowable Weight} = \text{Cargo Coef} \quad \text{Axle Weight} + \text{Intercept}$$

Equation 9. Allowable Weight Calculation

3.3.4 CB Lookup Table

MPALPTS uses a CB lookup table defining the acceptable ranges of CB given the total weight of all cargo. The actual acceptable CB lookup charts for both the C-17 and C-5 are described by extremely complex piecewise non-linear equations; however, the CB table MPALPTS uses is a discretised table used by load planners when calculations must be performed without the aid of AALPS (Air Mobility Command 2004). As in Roesener's work, this table is used to determine the acceptable and target CBs.

3.3.5 Pallet Placement Tables

The last table defines the pallet position locations and their respective weight and height constraints within each aircraft. MPALPTS uses this table to verify all the pallets loaded in the aircraft are in feasible locations.

3.4 Objective Function Costs

In order to drive the search toward improving feasible solutions, careful consideration must be made in choosing objective function costs. In MPALPTS, the total cost is divided into seven sub costs: Aircraft Usage Fee, Under Weight Fee, Over Weight Fee, CB Fee, Target CB Fee, Zone Fee and Ramp Fee. Each sub cost is multiplied by a

non-negative parameter allowing for fine adjustments of its contribution to the overall cost.

3.4.1 Aircraft Usage Fee

The aircraft usage fee is a user-definable parameter which assigns the entire fee if any cargo is loaded in a particular aircraft. Mathematically, it is represented as:

$$\sum_{j=1}^{A} C_j U_j,\ \mathrm{U}_j \in \{0,1\}\ \forall\ j = 1,2,...,A, C_j \in R$$

Equation 10. MPALP Usage Fee

where C_j represents the cost of the j^{th} aircraft and U_j is a binary variable indicating whether (U_j =1) or not (U_j =0) the j^{th} aircraft contains any cargo.

3.4.2 Under/Over Weight Fee

Unlike Roesener's weight penalties, MPALPTS uses three states to compute the cost of an aircraft's weight. Mathematically, they are:

$$\lambda_1 \sum_{j=1}^{A} \%ACL - 100\ ^2 U_j Y_j,\ \ Y_j, U_j \in \{0,1\}\ \forall\ \mathrm{j}=1,2,...,\mathrm{A}$$
$$\lambda_2 \sum_{j=1}^{A} 100 - \%ACL\, U_j (1 - Y_j) W_j,\ Y_j, U_j, W_j \in \{0,1\}\ \forall\ \mathrm{j}=1,2,...,\mathrm{A}$$
$$-\lambda_3 \sum_{j=1}^{A} 100 - \%ACL\, U_j (1 - Y_j)\ 1 - W_j\ ,\ Y_j, U_j, W_j \in \{0,1\}\ \forall\ \mathrm{j}=1,2,...,\mathrm{A}$$

Equation 11. Under/Over Weight Fee

where, Y_j is equal to one if the aircraft is overweight and zero otherwise, and W_j is equal to one if the aircraft is utilizing between 30% and 100% of its ACL and zero if the aircraft's ACL usage percentage is between 0% and 30%. These percentages were

determined through extensive testing using single factor at a time parameter setting techniques. Depending on the λ values, an example of this fee is graphically represented in Figure 6.

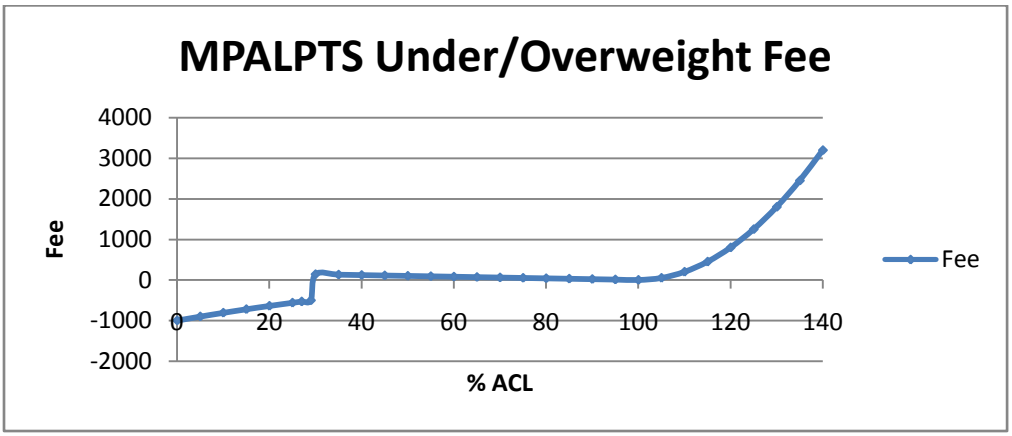


Figure 6. Representative Under/Overweight Curve

As an aircraft becomes overloaded, its cost becomes exponentially greater while solutions having a slight under-loading are only marginally penalized. However, if the MPALP heuristic is close to emptying an aircraft, it is encouraged to continue to do so with a negative cost (profit). This unique cost schedule drives solutions towards either 0% or 100% ACL utilization.

3.4.3 CB Fee and Target CB Fee

The goal of the MPALP search is to create feasible aircraft loadings whose CBs are as close as possible to the target CB. To accomplish this, the following longitudinal CB fee equations are used:

$$\lambda_4 \sum_{j=1}^{A} U_j B_j \ \text{TargetCB}_j\text{-CB}_j\ ^2, \ B_j, U_j = 0,1 \ \ \forall j = 1,...,A$$
$$\lambda_5 \sum_{j=1}^{A} U_j \ 1 - B_j \ \ \text{TargetCB}_j\text{-CB}_j\ ^2, \ B_j, U_j = 0,1 \ \ \forall j = 1,...,A, \lambda_4 >> \lambda_5$$

Equation 12. CB Fee

where B_j equals one if the j^{th} aircraft's CB is within acceptable CB limits and zero otherwise. U_j is a binary variable indicating whether (U_j = 1) or not (U_j = 0) the j^{th} aircraft contains any cargo. Notice that both equations assign fees based on the proximity of the calculated CB to the target CB; however, because $\lambda_4 >> \lambda_5$, the cost associated with being out of CB limits is much greater than being within CB limits.

3.4.4 Zone Fees

Zone fees are divided into five sub fees: (1) axles that, by themselves, are too heavy for a zone, (2) adjacent axles that are too heavy, (3) zones whose total cargo weight is too heavy, (4) zones having cargo items that require center loading but are not loaded in the center and (5) a pallet that is too heavy or tall for its pallet position. These five sub fees are not mutually exclusive which leads to the following complex additive fee calculation:

$$\lambda_6 \sum_{i=1}^{A} \sum_{j=1}^{2} \sum_{k=1}^{c_{ij}} \sum_{l=1}^{a_{ijk}} \left[H_{ijkl} \ \text{AxleWt}_{ijkl} - \text{MaxAxleWt}_{ijkl} \right] +$$

$$\lambda_6 \sum_{i=1}^{A} \sum_{k=1}^{c_{i1}} \sum_{l=1}^{a_{i1k}} \left[S_{i1kl} \ \text{AxleWt}_{i2kl} - \text{MaxAxleWt}_{i2kl} \right] +$$

$$\lambda_6 \sum_{i=1}^{A} \sum_{j=1}^{2} \sum_{k=1}^{c_{ij}} \left[T_{ijk} \ \text{CargoWt}_{ijk} \right] +$$

$$\lambda_6 \sum_{i=1}^{A} \sum_{m=1}^{Z} \sum_{j=1}^{2} \sum_{n=1}^{z_{ijm}} \left[D_{imjn} \ \text{CargoWt}_{imjn} - \text{MaxZoneWt}_{m} \right] +$$

$$\lambda_6 \sum_{i=1}^{A} \sum_{j=1}^{2} \sum_{k=1}^{c_{ij}} \left[P_{ijk} \left(\text{CargoWt}_{ijk} \right) \right]$$

$$H_{ijkl}, T_{ijk} \in \{0,1\} \ \forall \ i = 1,\ldots,A, \ j = \{1,2\}, k = 1 \ldots c_{ij}, l = 1 \ldots a_{ijk}$$

$$S_{i1kl} \in \{0,1\} \ \forall \ i = 1,\ldots,A, \ k = 1 \ldots c_{i1}, l = 1 \ldots v_{i1k}$$

$$D_{imjn} \in \{0,1\} \ \forall \ i = 1,\ldots,A, \ j = \{1,2\}, m = 1 \ldots Z, n = 1 \ldots z_{ijm}$$

$$P_{ijk} \in \{0,1\} \ \forall \ i = 1,\ldots,A, \ j = \{1,2\}, k = 1 \ldots c_{ij}$$

Equation 13. Zone Fees

Starting with the first term, c_{ij} represents the number of cargo items that are loaded in the j^{th} column of the i^{th} aircraft. Correspondingly, a_{ijk} represents the number of axles on the c_{ij}^{th} cargo item, and H_{ijkl} is a binary variable indicating whether ($H_{ijkl} = 1$) or not ($H_{ijkl} = 0$) the a_{ikj}^{th} axle is too heavy for its zone. If it is, it assigns a penalty of the difference between that axle's weight and the maximum allowed axle weight for that zone. The second term assigns fees when adjacent axles are too heavy. The term a_{i1k} (note: the middle index is a one, not *i*) represents the number of axles of the c_{ij}^{th} cargo item on the left side of the cargo compartment. S_{i1kl} (note: the second index is a one) is a binary variable indicating whether ($S_{i1kl} = 1$) or not ($S_{i1kl} = 0$) the a_{i1k}^{th} axle has an adjacency zone violation. If it does, it assigns a cost based on the difference between the conflicting axle's weight in the *right* column and the maximum allowable weight for that right side axle. This notation is used to avoid assigning double costs for a single adjacency violation. The third term represents the fee associated with a cargo item that should be

center loaded but is not. T_{ijk} is a binary variable which indicates whether (T_{ijk} = 1) or not (T_{ijk} = 0) the k^{th} item in column j on the ith aircraft should be centerline loaded. Items not properly centerline loaded are assigned a penalty equal to that cargo item's total weight. The fourth term determines if the total cargo weight in a particular zone is too heavy. The variable z_{imj} represents the total number of axles loaded in the j^{th} column of the i^{th} aircraft in zone m, and D_{imjn} is a binary variable indicating whether (D_{imjn} = 1) or not (D_{imjn} = 0) the total weight of the n axles in zone m is greater than the zones maximum allowable weight. If it is overloaded, the penalty assigned is equal to the difference between the total cargo weight in zone m and the maximum cargo weight allowed in zone m. Finally, P_{ijk} is a binary variable which indicates whether (P_{ijk} =1) or not (P_{ijk} =0) the k^{th} item in the jth column of the i^{th} aircraft is a pallet that is either too heavy or tall for its pallet position. The objective function is penalized by the pallet's weight if a violation is present.

Zone violations constitute infeasibilities which prevent safe flight; they are extremely difficult constraint types to model in the MPALP and can account for a large portion of the objective function value. When these violations are corrected, large objective function improvements are realized.

3.4.5 Ramp Fees

When an aircraft is being loaded, its ramp or ramps are lowered to allow rolling stock items to be towed or driven onto the aircraft; however, when an aircraft is in flight, its ramps are closed and lie at an upward angle with respect to the rest of the cargo floor. Because of this, one must consider the situation where a rolling stock item spans a ramp.

Ramp fees are assessed to prevent situations where a tracked vehicle spans a ramp or when a portion of any vehicle could contact the ramp. The only acceptable configuration for a wheeled vehicle to span the ramp is if it has at least one axle resting on the cargo floor and one resting on the ramp. This conservatively prevents the situation where all the vehicle's axles are on the cargo floor and the aft (or forward) portion of the vehicle overhangs a ramp. In this situation, the overhanging portion could come in contact with ramp when it is raised to the closed position. The following ramp fee equation is used:

$$\lambda_7 \sum_{i=1}^{A} \sum_{j=1}^{2} \sum_{k=1}^{c_{ij}} \left[R_{ijk} \ \text{CargoWt}_{ijk} \ \right]$$
$$R_{ijk} \in \ 0,1 \ \ \forall \ i = 1,...,A, \ j = 1,2, \ k = 1...c_{ij}$$

Equation 14. Ramp Fee

R_{ijk} is a binary variable which indicates whether (R_{ijk} = 1) or not (R_{ijk} = 0) the k^{th} item in column j on the i^{th} aircraft has a ramp constraint violation. If it is in violation, the assigned cost is a multiple of the item's weight.

3.5 MPALPTS Neighborhoods

The four neighborhood functions within MPALPTS are modeled after Harwig's and Roesener's work. They include *intra*-aircraft swap, *inter*-aircraft swap, *inter*-aircraft insert, and empty aircraft. In order to save computational time, only intra-aircraft swap actually evaluates its entire neighborhood. The other insert and swap neighborhoods evaluate a changing subset of their full neighborhood. Specifically, the first time the inter-aircraft swap neighborhood is called, it picks the best solution found from swapping

all cargo items between the following pairs of aircraft: $\left[\ a_1, a_2 \quad a_3, a_4 \quad a_5, a_6 \ \ldots \ a_A, a_1\ \right]$. The next time it is called, it increases the "distance" between aircraft and swaps all cargo items between the following pairs of aircraft: $\left[\ a_1, a_3 \quad a_2, a_4 \quad a_3, a_5 \ \ldots \ a_A, a_2\ \right]$. This leads to exactly *A* pairs of aircraft evaluated each time the neighborhood is called. Through algorithmic testing, this reduced neighborhood approach tended to find solutions of comparable quality to full neighborhood searches in significantly less time.

3.5.1 Inter-Aircraft Swaps

Inter-aircraft swaps are primarily used to rapidly improve the initial solution. The neighborhood explores all possible combinations of swapping one cargo item in one aircraft with another cargo item in another aircraft within the previously described reduced set of aircraft. To ensure pallets remain aft of rolling stock, pallets can only be swapped with other pallets, and rolling stock can only be swapped with rolling stock.

Unlike swapping pallets which always have identical width and length, rolling stock swaps are much more complicated. If the two cargo items are of different dimensions, the neighborhood function must determine if it is possible to fit a larger item in the location the smaller item previously occupied. If such a swap is not possible, then the algorithm determines if it is possible to slide the cargo to allow the larger item to fit. If the larger item still cannot fit, then that particular swap is ignored, and the algorithm continues to the next item. Thus, inter-aircraft swaps never allow the cargo items to exceed the physical dimensions of the cargo compartment; however, it is permitted to cause an aircraft to exceed its planning ACL. After each swap, both aircraft's loads are

sent through the Fix Load Function (section 3.6) which attempts to resolve zone, centerline and ramp violations. The best neighborhood solution is retained as the next solution.

3.5.2 Inter-Aircraft Inserts

Unlike inter-aircraft swaps, the inter-aircraft insert neighborhood excels at fine-tuning a solution with relatively small objective function improvements. When invoked, the inter-aircraft insert attempts to remove a cargo item from an aircraft and place it in a feasible location on a second aircraft. The algorithm attempts to place the cargo item in the farthest aft position of the column having the fewest number of cargo items while ensuring no rolling stock items are placed behind a pallet. If the cargo item will not fit, the inter-aircraft insert algorithm determines the maximum forward shift for the cargo in that column to accommodate the new item. If, even after sliding the cargo forward, the item will not fit, it performs the same procedure on the other column. If the item will not fit in either column, the insert is terminated and the algorithm continues to the next cargo item in the losing aircraft. Pallet items are only inserted when the gaining aircraft has open pallet positions. After each successful insert, the Fix Load Function is called. The intra-aircraft swap neighborhood is invoked on the best solution from the reduced neighborhood, and this solution is retained as the next solution.

3.5.3 Empty Aircraft Neighborhood

If MPALPTS determines it may be possible to empty an aircraft through the State Determination function (section 3.10) and it has already found a completely feasible

solution using the current set of aircraft, it invokes the empty aircraft neighborhood. This neighborhood, which is essentially a series of inter-aircraft inserts, identifies all the aircraft which currently utilize less than 75% of their ACL *or* less than 45% of their total space. These values were experimentally determined through single factor at a time testing. From this subset of aircraft, the neighborhood selects the aircraft with the smallest planning ACL to empty thereby leaving the bigger aircraft to hold the remaining cargo. It attempts to insert all of the cargo from this aircraft into any or all remaining aircraft. If the aircraft cannot be emptied, it removes that aircraft from the subset and repeats this procedure on the remaining aircraft in the subset until an aircraft is emptied or no aircraft remain in the subset.

3.5.4 Intra-Aircraft Swaps

Intra-Aircraft Swaps are intensification moves; their primary utility is to obtain an aircraft CB that is as close as possible to the target CB rather than explore new cargo permutations. This neighborhood is used to refine the initial solution as well as the best solution found in the inter-aircraft swap and insert neighborhoods. First, this function swaps all the items in a given column and computes the best solution. These swaps are relatively easy to accomplish because intra-column swaps cannot result in situations where the cargo does not fit in the column. Inter-column swaps are similar to inter-aircraft swaps in that if the cargo items are of different sizes, it may be possible for a particular swap to be infeasible. Thus, the same logic from *inter*-aircraft swaps is incorporated into these inter-column swaps within a single aircraft. The intra-aircraft swap also examines pallet swaps as well as inserting a pallet into an empty position aft of

rolling stock within the aircraft. After each successful swap, the Fix Load Function is called, and the swap with the lowest cost is chosen as the next solution.

3.6 Fix Load Function

The Fix Load Function is used to reload an aircraft if its current configuration has zone, ramp, axle or centerline violations. Essentially, it removes all of the rolling stock from the aircraft while leaving the pallets in their original locations. It replaces each rolling stock item, alternating between columns, in the original order and column from which it came; however, it attempts to arrange them such that there are no violations. This is done by placing the first item from the left side at the farthest forward point in the aircraft and then systematically sliding it aft until all violations are corrected. The Fix Load Function then places the first item from the right side and slides it in the same manner. Items continue to be inserted in alternating columns until all items have been repositioned or until an item overlaps a pallet or cannot feasibly fit in the remaining space. If a particular load cannot be corrected, the algorithm returns the original solution. After each successful fix, the algorithm calls the Slide CB function (section 3.7) to try to fix any CB violations.

3.7 Slide CB Function

After each successful Fix Load Function call, the Slide CB Function recalculates the aircraft's CB and determines whether it falls into acceptable ranges. If the CB is violated, it simultaneously slides all rolling stock items only enough to achieve a feasible CB. Because pallets have predetermined locations, they cannot be included in this

function. The required distance of the slide is simply the number of inches by which the current CB falls outside of the acceptable bounds. Mathematically, the CB for the i^{th} aircraft is defined by the cargo's total moments divided by the cargo's total weight where a cargo item's moment is the product of its weight and the FS on which its weight is centered.

$$CB_i = \frac{\sum_{j=1}^{2}\sum_{k=1}^{c_{ij}}\left[\ Weight_{ijk} \quad CB_{ijk}\ \right]}{\sum_{j=1}^{2}\sum_{k=1}^{c_{ij}} Weight_{ijk}} = \frac{\text{CargoMoment}_i}{\text{CargoWeight}_i}$$

Equation 15. CB Calculation

If CB_i is determined to be out of the range $CB_{\min}, CB_{\max}$, one can determine the distance the CB requires shifting by Equation 16 and the resulting CB can be computed using Equation 17.

$$\Delta = \begin{cases} \max\ CB_{\max} - CB_i, FwdShift_{\max} & CB_i > CB_{\max} \\ \min\ CB_{\min} - CB_i, AftShift_{\max} & CB_i < CB_{\min} \\ 0 & \text{otherwise} \end{cases}$$

Equation 16. CB Shift Calculation

$$CB_i + \Delta = \frac{\sum_{j=1}^{2}\sum_{k=1}^{c_{ij}}\left[\ Weight_{ijk} \quad CB_{ijk} + \Delta\ \right]}{\sum_{j=1}^{2}\sum_{k=1}^{c_{ij}} Weight_{ijk}}$$

Equation 17. Δ Calculation

In Equation 16, Δ represents the number of inches required to shift the cargo to the nearest acceptable CB boundary. $FwdShift_{\max}$ and $AftShift_{\max}$ represent the maximum amount the cargo can be shifted forward or aft within the longitudinal bounds of the cargo compartment; this equation includes any pallets present in the load. Using Equations 16 and 17 together, the amount one needs to shift the cargo itself to move the CB into its feasible bounds is simply Δ .

3.8 Tabu List

MPALPTS uses a single tabu list to track which moves are deemed to be tabu. The list, $TL_{a,c}$ $a = 1...A,\ c = 1...C$, is constructed as an $A \text{ x } C$ matrix and represents the iteration count when cargo item *c* can be returned to aircraft *a*. For example, assume the inter-aircraft insert neighborhood determined the best non-tabu solution resulted from moving cargo item 12 from aircraft 2 to some other aircraft. Any new solution which returns cargo item 12 to aircraft 2 is considered tabu until the tabu tenure expires. Mathematically, TL is updated by $TL_{2,12} = iteration + tabutenure$ where *iteration* is the iteration count when that particular move became tabu. An inter-aircraft swap, which can be described as two inter-aircraft inserts, makes two such updates to TL which prevents the swap.

To determine if a move is tabu, it is simply necessary to compare the current iteration count to the value within the appropriate TL cell; this is an extremely quick and efficient process. If the iteration count is greater than the tabu list's value, the move is

permitted. The inspiration for this method of tabu list construction came from Alfonsas Misevicius' tabu list in his traveling salesman problem heuristic (Misevicius 2004).

3.9 Initial Solution Generation

MPALPTS uses a heuristic to determine the number of aircraft required to load the entire cargo set and is roughly based off of Roesener's idea of dividing the cargo into groups (Roesener 2006).

First, the cargo is sorted by weight from largest to smallest and is then divided into two sets of k groups, one for pallets and one for rolling stock. Through experimentation, using $k = 4$ groups generated the highest quality solutions; this value coincides with Roesener's research (Roesener 2006). The initial solution generation heuristic begins with the first aircraft in the available aircraft list and attempts to load as many items as possible from the first (heaviest) rolling stock group into the aircraft. In order to keep the *number* of items on each side of the cargo compartment reasonably balanced, it loads the next item into the column with the least number of items; ties are arbitrarily assigned to the left column. If the first item of a group will not fit, the heuristic moves to the first item in the next group. It repeats this process until the first item in all four groups will not fit anywhere in the aircraft or until 100% of its ACL has been utilized, whichever occurs first. After an aircraft is filled with rolling stock, the same procedure is used to fill it with pallets. Once there are no open pallet positions, 100% of its ACL has been utilized or no other items will fit on the aircraft, the heuristic picks the next aircraft in the list and repeats the loading process until either all the cargo has been loaded or it runs out of available aircraft to load. If the user did not provide

enough aircraft to load all the cargo, the MPALPTS ends with an error message indicating it requires additional aircraft for the initial solution.

As the heuristic loads the aircraft with rolling stock, it determines the average axle weight and the item width which requires the item to be centerline loaded in the current aircraft. It uses these values to determine whether an item requires centerline loading; this generates a conservative number of aircraft in the initial solution.

After generating this initial solution, a series of intra-aircraft swaps and fix load procedures are performed on each aircraft in an attempt to find a feasible (or as close to feasible as possible) solution. However, the initial solution is only guaranteed to load all the available cargo such that it physically fits on each aircraft and the total weight of each aircraft does not exceed its ACL. It is up to the MALPTS to find the best feasible solution.

3.10 State Determination

MPALPTS assigns the state of the current solution into one of two categories. If MPALPTS determines that the possibility exists to empty one or more aircraft in the current solution and the current solution is feasible (i.e. no CB, zone, axle or ramp violations and no aircraft exceeds its ACL), then the system is in "State 1." If the system is not in "State 1", it is in "State 2."

In order to access the probability that MPALPTS can find a feasible solution using one less aircraft, the algorithm first determines the current solution's reduced excess weight by summing the differences of each aircraft's current weight and 90% of its ACL. As the average ACL of a solution approaches 100%, the search for feasibility

becomes very difficult because the space available to shift cargo is extremely limited. Using the reduced excess weight greatly improves the probability MPALPTS will be able to find a feasible solution after emptying an aircraft. This, in turn, reduces the computational time required for MPALPTS to search for a feasible solution using fewer aircraft when the probability of finding that solution is low. If the reduced excess weight is large enough to allow an aircraft to be emptied, the algorithm continues to the next step. Otherwise, it indicates a "State 2" solution and prohibits emptying an aircraft. In the next step, MPALPTS identifies which aircraft to attempt to empty. It does so by determining the set of aircraft currently using less than 75% of their ACL or 45% of their total space. If this set is nonempty, MPALPTS determines whether it is possible to transfer the weight of any of these aircraft to some or all of the remaining aircraft and returns the appropriate solution state.

3.11 MPALP Tabu Search Algorithm

Combining the actions described in the previous sections, the pseudo code for the MPALPTS is presented in Figure 7. A flowchart of the MATLAB functions can be found in Appendix B and a portion of the MATLAB code is contained in Appendix C. There are six main variables which control the search. First, the iteration counter tracks the number of iterations of the overarching while loop, and the search ends if it reaches 300 iterations. The trivial counter counts the number of trivially improving solutions; these are defined as an improvement of less than 10% from the previous solution. If the trivial counter reaches 50, the search terminates. These values were determined through

experimentally based single factor at a time analysis. To avoid premature termination, the trivial counter is reset to zero if an improving solution is significant (i.e. not trivial).

```
1.  Get as input (Cargo, Available Aircraft, CB Lookup Table, Zone Information, Pallet Information)
2.  Account for chain space and lateral space for each cargo item in Cargo
3.  Generate Initial Solution
        a. If there are not enough aircraft to load all the cargo, display error message, STOP.
4.  Calculate cost of Initial Solution
5.  Initialize FoundFeasibleSoln = 0  (1 if a feasible solution has been found, 0 otherwise)
6.  Initialize CannotEmptyUntil = 0  (Cannot try to empty an aircraft until iteration CannotEmptyUntil)
7.  Initialize InsertVSswap  = 0 ( All inter-aircraft swap moves)
8.  Create and Initialize tabu structure and Initialize all other variables
9.  While Iteration < 300 AND Trivial < 50 AND DisImprove < 50 loop
        a. Increment Iteration
        b. Determine State: 1 = Can Empty an acft, 2=Not state 1
        c. IF [(CannotEmptyUntil > Iteration) AND (State = 1)] then State = 2 END IF
        d. IF state = 1, Invoke Empy Aircraft procedure on current solution
            i. If successful
                1. Update Best Solution variables
                2. Update current solution
            ii. Increment CannotEmptyUntil by 10 iterations
        e. ELSE IF state = 2 AND insert neighborhood is the next move
            i. Invoke inter-acft insert neighborhood on current solution and return best non-tabu
               solution and updated tabu structure
                1. IF found an improved solution THEN
                        a. Update Best Solution variables
                        b. Update current solution
                   END IF
                2. Make returned solution the current solution
        f. ELSE IF state = 2 AND Swap neighborhood is the next move
            i. Invoke inter-acft swap neighborhood on current solution and return best non-tabu
               solution and updated tabu structure
                1. IF found an improved solution THEN
                        a. Update Best Solution variables
                        b. Update current solution
                   END IF
                2. Make returned solution the current solution
           END State if-then
        g. IF solution was a trivial improvement, then increment Trivial, Disimprove = 0
        h. ELSE IF solution was not improving, the increment Disimprove
        i. ELSE Trivial = 0, Disimprove = 0  (Found a significantly better solution)
           END IF solution…
        j. IF Disimprove + Trivial ≥ 10, InsVsSwap = -4
        k. ELSE InsVsSwap = 0
           END IF
    END While
10. Return Solution
```

Figure 7. MPALP Tabu Search Pseudo Code

The disimprove counter represents the number of sequential moves with a non-improving objection function values and is reset to zero if either a trivially or significantly better solution is found.

The next set of controlling variables dictate how the search progresses. First, it is critical to ensure the algorithm does not empty an aircraft before it has found a feasible solution using the current number of aircraft. If no feasible solution has been found on the current set of aircraft, there is no guarantee of feasibility after emptying one of the aircraft. Therefore, when MPALPTS has found a feasible solution using the current number of aircraft, it sets the binary variable "FoundFeasibleSoln" to 1; after this occurs, the algorithm is allowed to attempt to empty an aircraft. Similarly, if MPALPTS determines that it may be able to empty an aircraft and subsequently fails to empty an aircraft, it is likely that ensuing state calculations will result in trying to empty an aircraft again. Because of this, the "CannotEmptyUntil" variable is incremented by 10 each time the Empty Aircraft procedure is called regardless of success or failure. This prohibits the algorithm from trying to empty an aircraft again for at least 10 iterations. This value was shown, through experimentation, to allow sufficient changes in the solution to increase the probability of successfully emptying an aircraft.

The final variable, "InsertVsSwap" is designed to control the ratio of inter-aircraft insert to inter-aircraft swap neighborhood moves. For example, a value of negative three results in three insert moves for every swap move, and a value of zero results in exploring only swap moves. Swap moves generally produce large changes in the solution and are valuable in improving the initial solution; however, insert moves perform well at fine-

tuning the solution. Therefore, "InsertVsSwap" is initialized to zero and causes MPALPTS to refine the initial solution until stagnation occurs. This stagnation is recognized when the sum of the trivially improving and disimproving counters becomes greater than or equal to 10. Upon stagnation, "InsertVsSwap" is set to negative four and remains so until a feasible or a significantly improved solution is found at which time it is reset to zero.

If a feasible solution is found, the CannotEmptyUntil counter is incremented by 5 to allow MPALPTS to refine the feasible solution before trying to empty an aircraft. Additionally, the trivial and disimprove counters are adjusted to give MPALPTS a minimum of 20 iterations to attempt to empty the aircraft. These experimentally based values were shown to produce excellent solutions while also terminating the search sooner in the case an aircraft cannot be emptied.

After reaching a termination condition, MPALPTS returns up to three solutions and their associated costs: a feasible solution, a marginally infeasible solution, and a moderately infeasible solution. The three solutions allow the load planner to use MPALPTS as decision making tool to determine the costs of exceeding aircraft ACL versus benefits of potentially using fewer aircraft. Additionally, it exports the best feasible solution to an excel file which builds its visual representation. An example of the visual representation is located in Appendix D.

3.12 Robust Parameter Design

The goal of Robust Parameter Design (RPD) is to mathematically model a problem's solution space in terms of an algorithm's adjustable parameters to find settings

which produce high quality solutions across the spectrum of expected problem instances. Performing RPD tends to reduce the time spent on "randomly" adjusting parameters in hopes of finding robust settings.

The first step of RPD is to choose which parameters will be included within the mathematical models. Preliminary testing revealed the seven multipliers which define the cost of a particular solution were important to both solution quality and the time spent producing a solution. Table 4 shows the original parameter settings derived from the creation and informal testing of MPALPTS.

Table 4. Original Parameter Settings

Overweight Fee (λ_1)	Underweight Fee > 30 (λ_2)	Underweight Fee ≤ 30 (λ_3)	Target CB Fee (λ_4)	CB Fee (λ_5)	Zone Fee (λ_6)	Ramp Fee (λ_7)
1000	2	1	0.1	1000	0.25	1

The ranges in which each parameter is allowed to vary (Table 5) were chosen to contain the original settings. The upper bound on parameters affecting feasibility (λ_1, λ_5 , λ_6 , λ_7,) were set high relative to the remaining parameters in order to ensure RPD was able to find settings which drove the search to feasible regions of the solution space.

Table 5. Parameter Ranges

Parameter	Lower Bound	Upper Bound
λ_1	750	1250
λ_2	.1	20
λ_3	.1	20
λ_4	.1	20
λ_5	750	1250
λ_6	.1	1000
λ_7	.1	1000

3.12.1 Test Sets

Three pairs of problem instances were chosen as a representative sample of the types of loads MPALPTS may face. They include pallets only, rolling stock only and a mixed load of pallets and rolling stock. Each pair has one cargo set containing 75 items and one cargo set containing 200 items. The 75-item mixed set has 40 rolling stock items and 35 pallets while the 200-item mixed set has 90 rolling stock items and 110 pallets. These ratios of pallets to rolling stock items were purposefully chosen because MPALPTS has a more difficult time finding feasible solutions when the ratio is relatively equal. For test sets with pallets, individual pallets were chosen randomly among 30 sample pallets (Appendix A) whose weights were evenly spaced from 333 to 10,000 pounds. Pallet heights were chosen to be roughly commensurate to its weight such that lighter pallets are generally shorter than heavier ones. Rolling stock items were randomly picked from a set of 30 items selected from the extensive AALPS database (Appendix A). Table 6 summarizes the six test sets.

Table 6. Test Sets

Test Set	Pallets	Rolling Stock	Total
P75	75	0	75
P200	200	0	200
R75	0	75	75
R200	0	200	200
M75	30	45	75
M200	110	90	200

3.12.2 RPD Model Construction

The software package Design Expert was used to create, analyze and optimize a ¼ fractional central composite design which contained the seven design parameters along with a six-level categorical noise factor representing the test sets. It included 88 runs per test set for a total of 528 runs. MPALPTS was given an equal mix of C-5 and C-17 aircraft for each run. Response variables included the number of aircraft used in the best feasible solution and the time required to find the best feasible solution. If a particular combination of design parameters failed to produce a feasible solution, the number of aircraft returned was four times the number of aircraft in the initial solution, and the time required to complete the algorithm was also multiplied by four. This multiplication sufficiently separates feasible and infeasible responses. A quadratic regression model was built for each response variable as well as the response variable's variance.

A categorical noise factor complicates finding robust parameters. To find robust parameters that work for all test sets, Brenneman and Myers suggest optimizing the model for each test set and then using a binomial distribution to weigh each set of parameters based on their probability of occurrence (Brenneman and Myers 2003).

Because the probability of encountering any given test set is unknown, the categories were weighed based on their estimated relative difficulty. Rolling stock only problems have significantly more constraints than mixed and pallet only loads, and pallet only loads were found to be very resilient despite changes in the decision variables. Therefore, rolling stock only loads were weighted by 0.8, mixed loads were weighted by 0.15 and pallet only loads were weighted by 0.05. For the data collection phase of the experiment, tabu tenure was set at five and the trivially improving and disimproving counters were both set at 50.

3.12.3 Feasible Aircraft Model

A graphical representation of the quadratic aircraft model is shown in Figure 8. The model includes the CB Target Fee (λ_4), Zone Fee (λ_6), Ramp Fee (λ_7), the categorical variable and their quadratic interactions as the most significant components. The regression model itself was constructed using backward and manual regression techniques. The response variable was transformed using an inverse-square transformation to obtain normalized residuals and a better fitting model, so subsequent optimization required maximizing the transformed variable. The model has an adjusted R-squared value of 0.9256 and a signal to noise ratio of 55.64 indicating a reasonably good fit. The plots of the studentized residuals and predicted versus actual values both confirm a significant model.

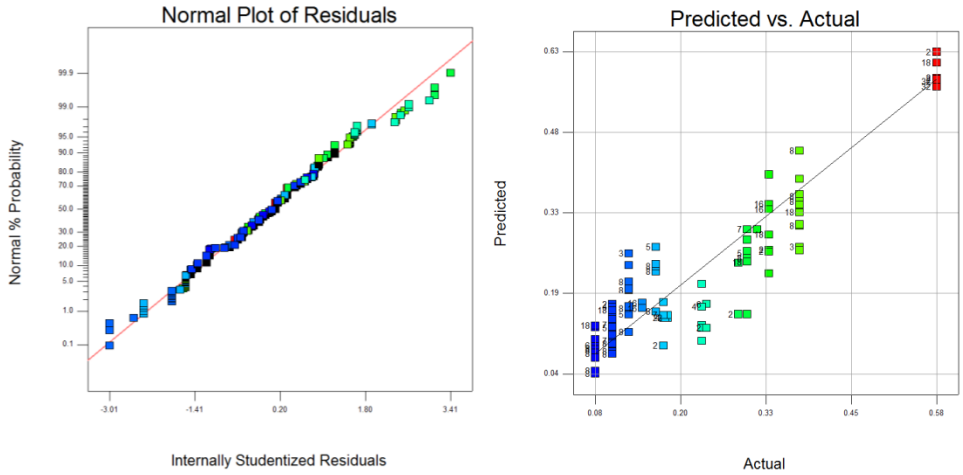


Figure 8. Aircraft Regression Model Plots

3.12.4 Time Model

The time model includes the same significant variables as the aircraft model, and the response variable was transformed using a natural log transformation. This model has an adjusted R-squared value of 0.9129 and a signal to noise ratio of 50.30 indicating a reasonably good fit. Figure 9 shows the plots of studentized residuals and predicted versus actual values both of which indicate a significant model.

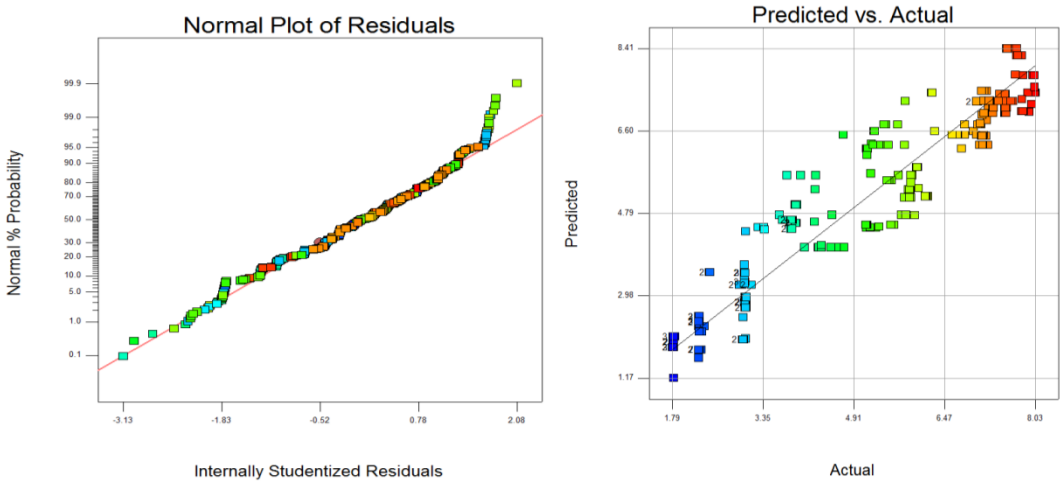


Figure 9. Time Regression Model Plots

3.12.5 RPD Results

To find the best parameter settings for each test case, both the time and aircraft models along with their associated variance models were simultaneously optimized. Six sets of optimal parameter settings (Table 7) were generated.

Table 7. Optimal Parameter Settings for Test Sets

Parameter	P75	P200	R75	R200	M75	M200
λ_1	772.78	1247.96	1124.47	772.78	1134.65	1145.39
λ_2	19.20	5.20	12.34	19.20	9.43	3.12
λ_3	19.99	19.96	4.59	19.99	12.45	8.22
λ_4	0.10	11.61	20.00	0.10	14.65	0.10
λ_5	750.00	1066.10	964.77	750.00	978.07	1154.95
λ_6	613.77	455.87	961.74	613.77	854.42	756.38
λ_7	475.20	497.59	872.24	475.20	701.64	554.98

Brenneman and Myers' binomial probability techniques work well when the optimal parameters are reasonably close to one another across the categorical variable; however, in this case, the settings for the target CB fee (λ_4) were grouped into two similar values. Using the previously mentioned weightings, the robust parameter settings are shown in Table 8.

Table 8. Robust Parameter Settings

λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7
1095.26	8.47	11.21	7.83	1016.76	788.35	630.30

Unfortunately, these settings resulted in very poor performance for test sets R200 and M200. Further testing revealed sets P200, R75 and M75 had only minor increases in required computational time when λ_4 was set to 0.1, so the three models were re-optimized in Design Expert using this setting for each level of the noise factor. This resulted in a similar situation for λ_2, so an additional iteration of re-optimizing had to be performed. The re-optimized and final robust parameters are shown in Tables 9 and 10 respectively.

Table 9. Re-Optimized Parameters

Parameter	P75	P200	R75	R200	M75	M200
λ_1	1000.00	751.57	782.68	1143.73	1177.36	1189.06
λ_2	2.00	2.00	2.00	2.00	2.00	2.00
λ_3	20.00	6.84	19.23	15.26	13.55	17.20
λ_4	0.10	0.10	0.10	0.10	0.10	0.10
λ_5	1000.00	765.56	797.87	866.22	1203.82	759.86
λ_6	500.05	494.57	872.33	475.58	702.88	554.93
λ_7	500.05	446.84	981.05	613.66	876.66	756.32

Table 10. Final Robust Parameters

λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7
1123.84	2.00	15.65	0.10	946.93	631.33	795.51

3.13 Summary

This chapter presented the details of MPALPTS including its four neighborhoods and its main controlling variables. The algorithm is designed to efficiently search for areas of high quality feasible solutions. As with any heuristic, finding the optimal variable settings for an algorithm is a critical step to obtain the best possible performance across the expected problem instances. Therefore, this chapter also presented robust parameter design techniques, as applied to MPALPTS, which illustrate an experimentally based method to find these optimal settings. The difficulties of categorical noise variables were also discussed

Chapter 4: Results

4.1 MPALPTS versus AALPS

To compare MPALPTS performance to AALPS, each test set was first loaded by AALPS using three different mixes of aircraft: an equal mix of C-5 and C-17 aircraft ("M" in Table 11), C-5 aircraft only (C-5), and C-17 aircraft only (C-17). AALPS has many loading options available to the user, so great care was taken to set both algorithms' adjustable parameters to equal settings. MPALPTS was given the exact number, mix and configuration of aircraft in the AALPS final solution for each test set. Table 11 summarizes the overall comparison between the AALPS and MPALPTS solutions. The table includes the percent ACL and space used for both methods. While the percent ACL calculation is straightforward, the space used is not necessarily intuitive, and it is unknown exactly how AALPS calculates this statistic; therefore, directly comparing the percentage of space used is not necessarily valid. MPALPTS defines an aircraft's total available space as the number of inches between the farthest forward and the farthest aft FS multiplied by the total cargo compartment width. If a cargo item is loaded only on one side of the cargo compartment, MPALPTS assumes it occupies the entire lateral space of that column from the item's forward FS to its aft FS including its required chain space. Similarly, center-loaded items are assumed to occupy the entire width of the aircraft floor.

Table 11. AALPS versus MPALPTS

Test Set			AALPS				MPALP-TS					
	Test	Min Acft	Mix C-5/ C-17	Acft	Avg % ACL	Avg % Space	Mix C-5/ C-17	Acft	Avg % ACL	Avg % Space	Time to Feas (sec)	Tot Time (sec)
P75	M	**3**	2/2	4	73.5	68.0	2/1	**3**	97.1	79.2	10.3	38.7
P75	C-5	**3**	3/0	**3**	83.7	71.3	3/0	3	83.7	63.4	14.2	44.8
P75	C-17	**5**	0/6	6	70.0	62.2	0/5	**5**	83.8	80.0	2.7	16.8
P200	M	**9**	5/5	10	83.0	78.2	5/4	**9**	92.9	78.1	29.2	114.7
P200	C-5	**7**	8/0	8	84.8	70.4	7/0	**7**	96.8	72.4	38.9	120.9
P200	C-17	**12**	0/14	14	88.7	72.9	0/12	**12**	98.9	88.8	305.7	363.6
R75	M	10	7/6	13	74.2	49.8	6/5	**11**	85.3	88.6	80.2	236.3
R75	C-5	8	13/0	13	69.9	44.8	9/0	**9**	85.4	88.8	180.7	202.2
R75	C-17	13	0/17	**17**	75.4	53.1	0/17	17	75.4	87.2	27.6	126.3
R200	M	27	17/17	34	80.0	49.7	15/14	**29**	93.6	88.1	1823.9	1902.
R200	C-5	22	26/0	26	80.4	46.7	24/0	**24**	92.4	88.5	1103.3	1187.
R200	C-17	37	0/44	**44**	82.3	52.4	0/44	44	82.3	85.0	587.7.8	1149.
M75	M	**7**	5/4	9	75.0	74.6	4/3	**7**	88.8	80.8	21.0	32.8
M75	C-5	**6**	7/0	7	73.9	54.0	6/0	**6**	86.3	84.6	83.9	96.4
M75	C-17	9	0/11	**11**	78.5	60.0	0/11	11	78.5	90.3	2.7	95.0
M200	M	**17**	10/10	20	80.0	57.1	9/8	**17**	93.3	87.2	382.8	426.1
M200	C-5	13	15/0	15	85.3	56.7	14/0	**14**	91.5	79.7	269.2	328.2
M200	C-17	17	0/25	**25**	85.4	59.7	0/25	25	85.4	89.1	192.3	620.2
S50	M	3	3/2	5	45.6	61.6	3/1	**4**	51.6	86.2	43.2	53.5
M800	M	75	45/44	89	89.0	56.5	43/42	**85**	93.0	78.6	5801.8	1585

The number of aircraft required by both algorithms is also reported, as is the theoretical minimum number of aircraft required for a feasible solution. This theoretical minimum is computed by individually subtracting the ACL of each aircraft given to MPALPTS from the total weight of the cargo until all the cargo weight is “loaded” onto the aircraft. This minimum does not account for any cargo constraints and therefore may not represent the actual optimal number of aircraft for any given cargo set. The

theoretical limit can, however, be considered an absolute lower bound for feasible solutions.

The time required to reach the solution is only provided for MPALPTS; AALPS produces solutions almost instantaneously. The time necessary to reach a feasible solution using the least number of aircraft is represented in the "Time to Feasible" column while the overall running time of the algorithm is represented in the "Overall Time" column. The "Overall Time" column can include time expended searching for a feasible solution using fewer aircraft than the best feasible solution found, and it may include time spent refining the final solution. Two additional test problems are included: one with 50 of the same rolling stock item (S50) and one with 400 pallets and 400 rolling stock items (M800). These test problems are designed to illustrate other possible types of cargo loading outside of the original RPD models. Each test set for MPALPTS was executed on an Intel Centrino dual-core processor laptop running at 2.4 GHz with 3 GB of memory.

While RPD is extremely useful in finding robust parameter settings, it also tends to sacrifice excellent solution quality in individual test problems for adequate solution quality across all test problems. For example, some parameter settings found feasible solutions for R200 C-17 using 43 aircraft. Additionally, MPALPTS found a marginally infeasible solution to M800 using 84 aircraft with two of the aircraft being overloaded at 100.15% and 100.002% of their ACL, respectively.

Both MPALPTS and AALPS used the same number of aircraft in five test sets (P75 using only C-5 aircraft, R200, M75, R75, and M200 using only C-17 aircraft).

MPALPTS required fewer aircraft than AALPS in 15 tests sets; MPALPTS achieved the absolute lower bound for feasible solutions in 9 test problems. On average, MPALPTS used 11.48% fewer aircraft than AALPS with a maximum percentage of 30.77% fewer (R-75 C-5 only); however, these improved solutions required more computational time than AALPS.

The most difficult test problem of the original six was R200; this scenario had the largest number of axle, centerline and zone constraints of the six test sets. MPALPTS required approximately 32 minutes to complete the mixed C-5/C-17 test case. The M800 test set, which contains 9.5 million pounds of cargo, required approximately 270 minutes to complete; MPALPTS found a feasible solution in about 91 minutes, and the remaining time was spent locating the aforementioned marginally infeasible solution. While this appears to be a relatively long time (compared to AALPS), MPALPTS saved 4 aircraft over AALPS; a highly trained loadmaster would have required several days or weeks to find a similar solution. Appendix E contains specific results for each test problem including the number of items loaded on each aircraft and the percentage of ACL and space used.

4.2 Load Validation

In order to validate MPALPTS results, a sample of its load plans were manually recreated in AALPS which, in turn, displays any constraint violations present. The only adjustments required to MPALPTS solutions were related to the C-5 crew and troop compartment ladders which AALPS assumes are in the “down” position. MPALPTS assumes the height of each item will not impact any portion of the aircraft protruding

from the ceiling of the cargo compartment. In each case tested, sliding the cargo item laterally corrected the violation and did not require any items to be removed from the aircraft. Additionally, AALPS rounds the CB of each item to the nearest whole number prior to making CB calculations. Despite this fact, the CB calculated by AALPS was always within 1.5 inches of the MPALPTS calculated CB. Appendix F has two load plans from AALPS which represent the feasibility of MPALPTS solutions.

4.3 Applied Results

From February 2007 to January 2008, AMC reportedly flew 686 C-5 and 1551 C-17 multi-leg operational missions (Anderson 2008). Assuming all missions were originally planned with AALPS and were reloaded with MPALPTS (which, on average, increased airlift efficiency by 11.48% over AALPS), AMC would have flown 75 fewer C-5 missions and 171 fewer C-17 missions. If these airlift missions averaged 30 flight hours from the Continental United States to the cargo's destination and back, AMC would have saved $117,978,930 in this twelve-month period (using the previously mentioned hourly flight costs for C-5 and C-17 aircraft). In less than nine years of using MPALPTS, AMC could realize airlift savings of over one billion dollars.

While MPALPTS takes significantly longer than AALPS to find solutions, its ability to feasibly load a given set of cargo using fewer aircraft than AALPS would significantly improve the USAF's ability to efficiently utilize its airlift fleet. This increased efficiency could result in significant cost reductions for Air Mobility Command.

Chapter 5: Future Research

Despite MPALPTS successes, there are many aspects of this difficult problem which merit future study. First, all the cargo loaded with MPALPTS are assumed to be destined for the same location. In reality, this is rarely the case; including the destination of individual pieces of cargo as well as the planned stops of each aircraft would be required to better model this problem. Items should be positioned within the cargo compartment to facilitate efficient offload at each location an aircraft transits. This type of problem would be a pick-up and delivery bin packing problem. Second, MPALPTS assumes no hazardous cargo is present. Hazardous cargo must be separated by a specific distance within an aircraft, and some types are not allowed to be transported on the same aircraft. This constraint could be relatively easily modeled in MPALPTS by adding additional cost and feasibility requirements. Third, one of the greatest challenges to modeling airlift is handling large, oddly shaped cargo items (such as helicopters). To efficiently load these types of cargo, they must be rotated within the cargo compartment; therefore, they present an added level of complexity to the overall problem which could be explored. Fourth, the MPALPTS assumes rolling stock items will not contact the ceiling of the cargo compartment regardless of their position. Accounting for available space within an aircraft in three dimensions is also a difficult problem because there are many obstructions (such as the C-17's center fuel tank or the C-5's aircraft ladders) which limit the allowable height of a cargo item. Resolving these additional problem constraints would create a more robust and operationally useful product. Additionally,

cargo zones have pounds per square inch limitations for the tires or vehicle tracks. MPALPTS assumes these constraints are satisfied or would be satisfied by adding shoring. Explicitly defining these constraints would add validity to the model. Finally, as with any new algorithm, improving MPALPTS in terms of its solution quality or run time would also be a useful endeavor.

Appendix A: Test Set Cargo

A.1 Rolling Stock

Table 12. Rolling Stock Cargo

Length	Width	Height	# Axles	Axle Loc	2	3	4	5	6	Axle Wts	2	3	4	5	6	Tot Wt	CB	Cargo Num	Type (1=RS)	Type	NSN
162																2531	99	1	1	TRAILER TANK WATER	
294																8000	160	2	1	TRAILER FLATBED	
147																1340	103	3	1	TRAILER CARGO 3/4-TON	
258																6220	107	4	1	TRUCK, 6 PAX, 4X4	
209																2900	91	5	1	TRUCK PICKUP 4200	
146																8730	98	6	1	TRUCK FORK LIFT	
223																8170	124	7	1	TRUCK CARGO TACTICAL	
266																22146	144	8	1	TRUCK DUMP 5-TON	
401																38800	210	9	1	TRUCK CARGO 10T 8X8	
191																5600	88	10	1	TRK, UTIL, HVY, 2 1/4T, HMMWV	
180																5280	95	11	1	TRK, UTIL, CARGO/TRP CARR, 1 1/4T, W/EQP, HMMWV	
250																15920	151	12	1	TRACTOR, ALL-WHL-DRIVE, W/ATTACHMENTS	
204																7500	95	13	1	TRK, AMBUL, 4-LTR, ARMD, 2 1/4T, HMMWV	
265																45080	120	14	1	CARRIER AMMO TRACKED VEH	
255																12160	103	15	1	LATRINE SVC TRK	
191																8400	92	16	1	EXPL ORD DISP TRK MTD	
150																3050	112	17	1	TRAILER CABLE REEL	
119																765	72	18	1	TRAILER PLATFORM WHS	
315																35975	174	19	1	MRAP BAE-TVS CAT II	
227																2520	130	20	1	TRAILER BASIC UTILITY	
122																2140	51	21	1	PUMP, WATER, 350 GPM	
420																50570	196	16	1	TRK CGO HVY PLS TRANS	
401																55665	198	23	1	TRK TANK 2500 GAL	
269																15760	149	24	1	TRK VAN SHOP 2-1/2-T	
190																23500	92	25	1	COMBAT VEH IMP TOW TRACKED	
294																41160	143	26	1	ANTI-TANK VEH/STRYKER	
252																27650	132	27	1	LAV, ANTI-TANK	
108																620	63	28	1	TRAILER CARGO 1/4-TON	
172																4120	105	29	1	TRAILER VAN SHOP	
137																3500	79	30	1	CHASSIS TRAILER	

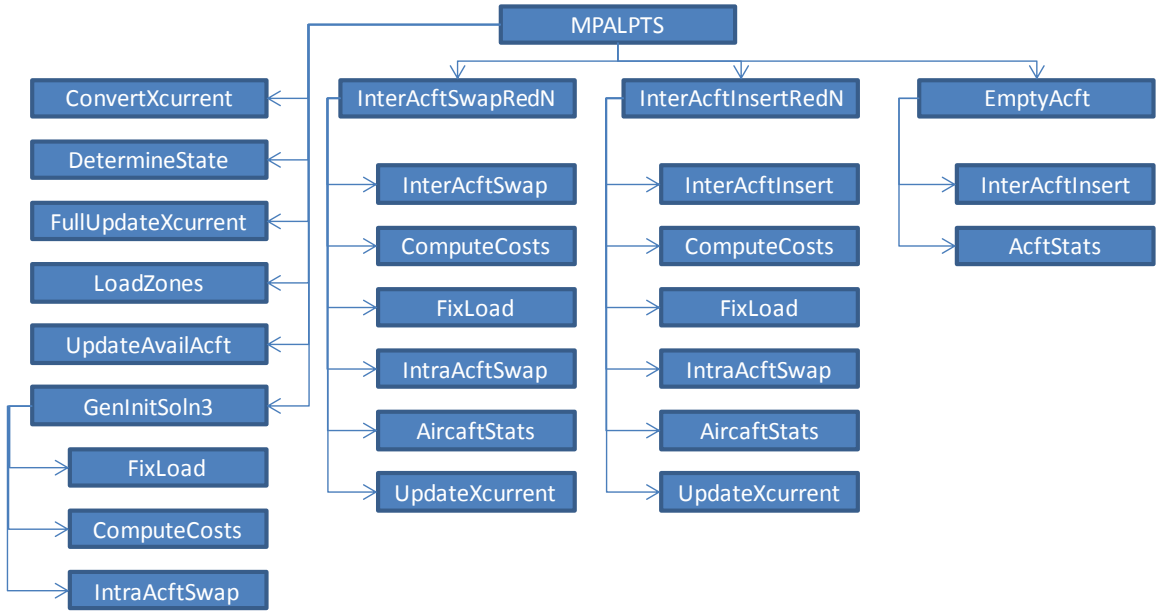
A.2 Pallets

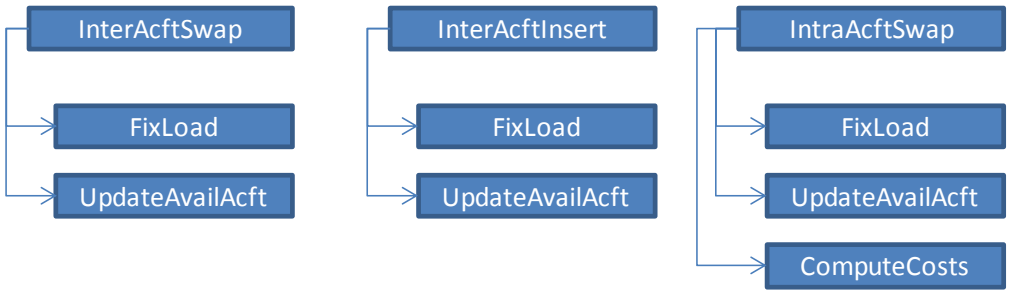
Table 13. Palletized Cargo

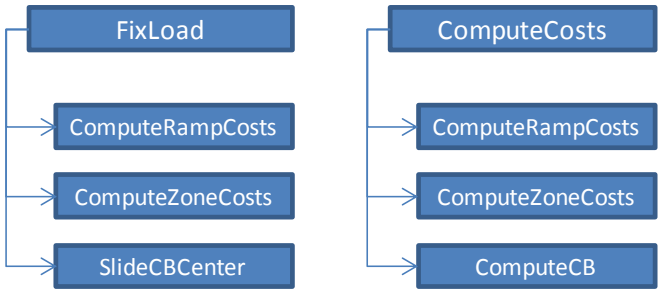
Length	Width	Height	# Axles	Axle Loc	2	3	4	5	6	Axle Wts	2	3	4	5	6	Tot Wt	CB	Cargo Num	Type (1=RS)	Type
88	108	24	0	0	0	0	0	0	0	0	0	0	0	0	0	333	0	1	2	Pallet
88	108	30	0	0	0	0	0	0	0	0	0	0	0	0	0	667	0	2	2	Pallet
88	108	38	0	0	0	0	0	0	0	0	0	0	0	0	0	1000	0	3	2	Pallet
88	108	27	0	0	0	0	0	0	0	0	0	0	0	0	0	1333	0	4	2	Pallet
88	108	33	0	0	0	0	0	0	0	0	0	0	0	0	0	1667	0	5	2	Pallet
88	108	35	0	0	0	0	0	0	0	0	0	0	0	0	0	2000	0	6	2	Pallet
88	108	41	0	0	0	0	0	0	0	0	0	0	0	0	0	2333	0	7	2	Pallet
88	108	45	0	0	0	0	0	0	0	0	0	0	0	0	0	2667	0	8	2	Pallet
88	108	43	0	0	0	0	0	0	0	0	0	0	0	0	0	3000	0	9	2	Pallet
88	108	50	0	0	0	0	0	0	0	0	0	0	0	0	0	3333	0	10	2	Pallet
88	108	48	0	0	0	0	0	0	0	0	0	0	0	0	0	3667	0	11	2	Pallet
88	108	48	0	0	0	0	0	0	0	0	0	0	0	0	0	4000	0	12	2	Pallet
88	108	55	0	0	0	0	0	0	0	0	0	0	0	0	0	4333	0	13	2	Pallet
88	108	37	0	0	0	0	0	0	0	0	0	0	0	0	0	4667	0	14	2	Pallet
88	108	43	0	0	0	0	0	0	0	0	0	0	0	0	0	5000	0	15	2	Pallet
88	108	61	0	0	0	0	0	0	0	0	0	0	0	0	0	5333	0	16	2	Pallet
88	108	77	0	0	0	0	0	0	0	0	0	0	0	0	0	5667	0	17	2	Pallet
88	108	82	0	0	0	0	0	0	0	0	0	0	0	0	0	6000	0	18	2	Pallet
88	108	73	0	0	0	0	0	0	0	0	0	0	0	0	0	6333	0	19	2	Pallet
88	108	45	0	0	0	0	0	0	0	0	0	0	0	0	0	6667	0	20	2	Pallet
88	108	95	0	0	0	0	0	0	0	0	0	0	0	0	0	7000	0	21	2	Pallet
88	108	83	0	0	0	0	0	0	0	0	0	0	0	0	0	7333	0	22	2	Pallet
88	108	72	0	0	0	0	0	0	0	0	0	0	0	0	0	7667	0	23	2	Pallet
88	108	84	0	0	0	0	0	0	0	0	0	0	0	0	0	8000	0	24	2	Pallet
88	108	78	0	0	0	0	0	0	0	0	0	0	0	0	0	8333	0	25	2	Pallet
88	108	76	0	0	0	0	0	0	0	0	0	0	0	0	0	8667	0	26	2	Pallet
88	108	81	0	0	0	0	0	0	0	0	0	0	0	0	0	9000	0	27	2	Pallet
88	108	80	0	0	0	0	0	0	0	0	0	0	0	0	0	9333	0	28	2	Pallet
88	108	64	0	0	0	0	0	0	0	0	0	0	0	0	0	9667	0	29	2	Pallet
88	108	90	0	0	0	0	0	0	0	0	0	0	0	0	0	10000	0	30	2	Pallet

Appendix B: MATLAB Flowchart

The following figures illustrate the overall architecture of the MPALPTS code.







Appendix C: MATLAB Code

Only the code for the MPALPTS main procedure and the Fix Load procedure are included. Complete electronic copies of the code can be obtained by emailing the author at robert.nance@us.af.mil.

```
function
[ZbestFeas,XbestFeas,ZbestMargInfeas,XbestMargInfeas,ZbestModInfeas,...

XbestModInfeas,AvailAcftFeas,AvailAcftMargInfeas,AvailAcftModInfeas,t,.
..
  tottime] = MPALPTS (file,Cargo,AvailAcft,CBTable...
  ,A1ZT,A2ZT,A1PT,A2PT)

%*************************************************************************
% Procedure:  MPALPTS
% Author   : Maj R. Larry Nance
% Purpose  : Runs the tabu search for finding feasible loadings for aircraft based
%             on the inputs and returns the best feasible solution it found along with some
%            statistics about how long the solution took, etc.
% Inputs:   file :    This is a text field that points to where the excel file is
%                     that contains all the tables defining the variables.  If the
%                    variables are known already, I just pass in 'file' or any
%                    other string.  If the variables are passed in, the procedure
%                    skips trying to open the file.  If some of the variables are
%                    missing, it will go to the file and ask you to input the
%                    appropriate files
%         Cargo:    A n x 20 matrix where n is the number of cargo items.
%                    Columns are defined in the excel file "Pallet Testing"
%         AvailAcft:a x 19 matrix which defines various aspects of the
%                    aircraft availiable for loading (a = number of acft)
%         CBTable  :Lookup table for the CB
%         A1ZT      :Aircraft 1 Zone Table -- defines zone positions and
%                    constraints
%         A2ZT     :Aircraft 2 Zone Table
%         A1PT     :Aircraft 1 Pallet Table -- defines pallet positions and
%                    constraints
%         A2PT     :Aircraft 2 Pallet Table
% Outputs:  ZbestFeas           :Best objective function value of best feasible solution
%             XbestFeas           :Representation of best feasible solution found
%             ZbestMargInfeas :Best cost of best marginally
%                                  infeasible solution (ACL > 100 and <= 102.5)
%             XbestMargInfeas :Representation of best marginally infeasible solution
%             ZbestModInfeas  :Best objective function value of best moderately
%                                  infeasible solution (ACL >102.5, < 105)
%             AvailAcft          :Best feasible, marginally infeasible and moderately
```

```
%                                infeasible representations of the set of aircraft
%                                found during search
%              t                :Time it took to reach best feasible solution
%         tottime               :Time it took for the whole tabu search to run
% CalledBy: User (no other procedures call this.  This is the main procedure
% Calls   : ChainSpace, GenInitSoln3, LoadZones, InterAcftInsertRedN,
%         InterAcftSwapRedN, ConvertXcurrent, DetermineState
%*************************************************************************

%Robust parameter settings found by RPD.
%Costs = [MaxTrivial, MaxDisimprove, InsertVsSwap, Tabu Tenure, Overweight Fee,
%        Underweight Fee < 30, Underweight Fee >= 30, CB Target Fee, CB Fee,
%        Ramp Fee, Zone Fee]

Costs = [50.00  50.00 0.00 5.00 1123.84 2.00 15.65 0.10 946.93 795.51 631.33];


%Starting Time
starttime = cputime;

%*************************************************************************
%Allow manual input option for data
%*************************************************************************
if nargin == 1
  disp ('Input Cargo File from Excel Sorted by Cargo ID.')
  Cargo = xlsread (file,-1);
  disp ('Input Available Aircraft File from Excel Already Sorted by priority.')
  AvailAcft = xlsread (file,-1);
  disp ('Input Zone table for aircraft type 1.')
  A1ZT = xlsread (file,-1);
  disp ('Input Zone table for aicraft type 2.')
  A2ZT = xlsread (file,-1);
  disp ('Input CB Lookup Table')
  CBTable = xlsread (file,-1);
  disp ('Input pallet table for aircraft type 1.')
  A1PT = xlsread (file,-1);
  disp ('Input pallet table for aircraft type 2.')
  A2PT = xlsread (file,-1);

end
%Calculate min theoretical aircraft using the order of aircraft in AvailAcft
TotCargoWt = sum (Cargo(:,17));
iter = 0;
while TotCargoWt > 0
  iter = iter + 1;
  TotCargoWt = TotCargoWt - AvailAcft(iter,2);
end
MinAcftTheoretical = iter;
disp (['Minimum theoretical aircraft based on ACL is ' num2str(MinAcftTheoretical) '.']);

%Load Zones
[Zones] = LoadZones (A1ZT,A2ZT);
```

```
%Create CB Lookup Table
CBLookup (1,:,:) = CBTable (:,2:4);
CBLookup (2,:,:) = CBTable (:,6:8);

%Create Pallet Table which defines the pallets for each aircraft
A1PTRows = size (A1PT,1);
A2PTRows = size (A2PT,1);
PalletTable = zeros (max (A1PTRows,A2PTRows),5);
PalletTable (1,1:A1PTRows,1:5) = A1PT;
PalletTable (2,1:A2PTRows,1:5) = A2PT;

% Fix the length of each cargo item to acocunt for chaining space
[Cargo] = ChainSpace (Cargo);

%Generate Initial Solution
disp ('Generating Initial Solution')
[Xcurrent,AvailAcft,success,feasible,Zbest] = GenInitSoln3 (Cargo,...
  AvailAcft,Zones,CBLookup,PalletTable,4,Costs);

if success ==1 %If I came up with a initial solution

  %Number of aircraft in initial solution
  NumAcft = size (Xcurrent,1);



  %Initialize all the variables
  TrivialImproveMove = 0;
  DisImproveMove = 0;
  InterSpanInsert = 1; %used to spread out the reduced insert algorithm
  InterSpanSwap = 1;
  iteration = 0;
  Zcurrent = Zbest;
  ZbestFeas = inf;
  ZbestMargInfeas =inf;
  ZbestModInfeas = inf;
  XbestFeas = 0;
  XbestMargInfeas = 0;
  XbestModInfeas = 0;
  AvailAcftFeas = AvailAcft;
  Xbest = Xcurrent;
  AvailAcftBest = AvailAcft;
  AvailAcftMargInfeas = 0;
  AvailAcftModInfeas = 0;
  tabulist = zeros (size(AvailAcft,1),size(Cargo,1));
  %tabutenure = Costs(4);
  tabutenure = 5;
  foundfeas = 0;  %= 1 if we have found feasible soln using NumAcft aircraft
  MaxTrivial = Costs(1);
  MaxDisImp = Costs(2);
```

```
MaxIter = 300;

%Ratio of Insert and Swap nieghborhoods explored
%Neg number means do more inserts than swaps
%Pos number means to more swaps than inserts
%InsVsSwap = Costs(3);
InsVsSwap = 0;

ZoneMult = Costs(11);
RampMult = Costs(10);

%*************************************************************************
%Start Tabu Search
%*************************************************************************
%Step 1:  Determine what state we are in
%   State 1:  Have found a feasible solution AND algorithm thinks emptying
%             is possible
%   State 2:  May or may not have found feasible solution.  Continuing to
%   refine search through inter and intra aircraft swaps/inserts
%
%*************************************************************************

CannotEmpty = 0; %= number of iterations must wait until state 1 to refine soln
while TrivialImproveMove < MaxTrivial &&...
   DisImproveMove < MaxDisImp && iteration < MaxIter
  iteration = iteration + 1;

  %Display current iteration counters
  text = sprintf ('%8s\t%8s\t%8s\t%8s\t%8s','Trivial','Disimp', 'Iter',  'NumAcft',  'InsVsSwap');
  disp(text)
  text = sprintf('%8.0f\t%8.0f\t%8.0f\t%8.0f\t%8.1f',TrivialImproveMove, DisImproveMove, iteration,...
    NumAcft, InsVsSwap);
  disp (text)

  %Determine the current solution's state
  [state] = DetermineState (AvailAcft,foundfeas);


  %if unable to empty acft due to space restrictions, cannot try again
  %until CannotEmpty < iteration
  if CannotEmpty > iteration && state == 1
    state = 2;
  end

  %*************************************************************************
  %Empty an Aircraft
  %*************************************************************************
  % If state equal one, then try to empty an aircraft
  %*************************************************************************
  if state == 1
    disp ('Attempting to Empty Acft')
```

```
[TempXcurrent,TempAvailAcft,worked] = EmptyAcft (Xcurrent,...
  AvailAcft,Cargo,CBLookup,Zones,PalletTable,ZoneMult,RampMult);
%Check to see if TempXcurrent has any errors
%  [good] = CheckXcurrent2 (TempXcurrent,TempAvailAcft,Cargo,PalletTable);
%  if good == 0
%    disp ('Error EmptyAcft');
%  end
if worked == 1
  foundfeas = 0; %assume solution is not feasible
  Zcurrent = inf; %best cost found from emptying aircraft
  Xcurrent = TempXcurrent; %best solution found from emptying aircraft
  AvailAcft = TempAvailAcft;
  DisImproveMove = 0;  %Reset DisImprove moves to 0 to allow for more refining.
  Zbest = inf;  %Reset the search because we have now reduced the aircraft by one
  ZbestFeas = inf;
  ZbestMargInfeas = inf;
  ZbestModInfeas = inf;
  tabulist = zeros (size(AvailAcft,1),size(Cargo,1)); %reset tabu list
  disp ('Reduced number of Acft by one!!!')
  %TEST
  for acft = 1:size (Xcurrent,1)
    [Xcurrent] = FixLoad (Xcurrent,acft,AvailAcft,Cargo,...
      CBLookup,Zones,PalletTable,ZoneMult,RampMult);
  end
else %if worked ~= 1-->Can't empty acft
  disp ('Cannot Empty Acft')
  CannotEmpty = iteration + 2;
end


  %Logic to figure out if we do an insert or swap neighborhood next
elseif state == 2 && ((InsVsSwap > 0 && NumSwaps >= InsVsSwap) || ...
  (InsVsSwap < 0 && NumInserts < abs(InsVsSwap)))
 if InsVsSwap > 0
  NumSwaps = 0;
 else
  NumInserts = NumInserts + 1;
 end

 %*************************************************************************
 %Inter-Aircraft Insert Reduced Neighborhood
 %*************************************************************************

 %Explore the INTER acft insert neighborhood.  Generate a solution.  If it is not
 %tabu, then allow it. (The InterAcftInsertRedN determines if a soln is
 %tabu or not

 %if insertspan > NumAcft-1, then tries to insert to itself
 if InterSpanInsert >= NumAcft -1
  InterSpanInsert = 1;
 end
 disp('Inter Aircraft Insert Reduced Neighborhood.....')
```

```
[tabulist,TempXcurrent,TempAvailAcft,TempCost,feasible] = InterAcftInsertRedN...
  (tabulist,iteration,tabutenure,AvailAcft,Xcurrent,Cargo,CBLookup,Zones,...
  InterSpanInsert,PalletTable,Costs);

InterSpanInsert = InterSpanInsert + 1;

%[good] = CheckXcurrent2 (TempXcurrent,TempAvailAcft,Cargo,PalletTable);
%if good == 0
%  disp ('issues after InsertRedNeigh')
%end

%if this is the first time we have found a feasible solution, give it at
%least 3 iterations to improve it before we try to empty an acft
if feasible == 1 && foundfeas == 0
  CannotEmpty = iteration + 5; %give some time to improve on the feasible soln
  feastime = cputime - starttime;
  text = sprintf ('%1s%1f%1s%d%s','**********Found feasible solution in ',...
    feastime, ' seconds using ',NumAcft,' aircraft.');
  disp(text)
  beep

  %Reset Counters to allow enough time to improve solution and then, if
  %possible, empty an aircraft and continue on.  If we empty an aircraft,
  %then we will reset the counters to zero
  %if we can't empty the aircraft, the algorithm will quit
  TrivialImproveMove = max (0,MaxTrivial - 20);
  DisImproveMove = max(0,MaxDisImp - 20);
end

%if I found a feasible solution and it is better than the current feasible
%solution
if feasible==1 && TempCost < ZbestFeas
  if TempCost < Zbest
    Zbest = TempCost;
  end
  XbestFeas = TempXcurrent;
  ZbestFeas = TempCost;
  AvailAcftFeas = TempAvailAcft;
  foundfeas = 1;

  %ELSEIf I found a marginally infeasible solution and it is better than current
  %marginally infeasible solution
elseif feasible == 2 && TempCost < ZbestMargInfeas
  XbestMargInfeas = TempXcurrent;
  ZbestMargInfeas = TempCost;
  AvailAcftMargInfeas = TempAvailAcft;

  %ELSE If I found a moderately infeasible solution and it is better than current
  %moderately infeasible solution
elseif feasible==3 &&TempCost < ZbestModInfeas
  XbestModInfeas = TempXcurrent;
  ZbestModInfeas = TempCost;
```

```
  AvailAcftModInfeas = TempAvailAcft;
 end

 %Best solution out of neghborhood becomes current solution
 Zcurrent = TempCost;
 Xcurrent = TempXcurrent;
 AvailAcft = TempAvailAcft;


 %*************************************************************************
 %Inter-Aircraft Swap Reduced Neighborhood
 %*************************************************************************

else
 %Update counters to figure out if we do a insert or swap next
 if InsVsSwap > 0
  NumSwaps = NumSwaps + 1;
 else
  NumInserts = 0;
 end

 if InterSpanSwap >= NumAcft-1  %if insertspan > NumAcft, then tries to insert to itself
  InterSpanSwap = 1;
 end

 disp('Inter Aircraft Swap Reduced Neighborhood...')
 [tabulist,TempXcurrent,TempAvailAcft,TempCost,feasible]= InterAcftSwapRedN...
  (tabulist,iteration,tabutenure,AvailAcft,Xcurrent,Cargo,CBLookup,...
  Zones,InterSpanSwap,PalletTable,Costs);
 InterSpanSwap= InterSpanSwap + 1;

 %[good] = CheckXcurrent2 (TempXcurrent,TempAvailAcft,Cargo,PalletTable);
 %if good == 0
 %  disp ('issues after Swap Neigh')
 %end

 %if this is the first time we have found a feasible solution, give it at
 %least 3 iterations to improve it before we try to empty an acft
 if feasible == 1 && foundfeas == 0
  CannotEmpty = iteration + 5;%give some time to improve on the feasible soln
  feastime = cputime - starttime;
  text = sprintf ('%1s%1f%1s','Found feasible solution in ', feastime, ...
   ' seconds.');
  disp(text)
  beep

  %Reset Counters to allow enough time to improve solution and then, if
  %possible, empty an aircraft and continue on.  If we empty an aircraft,
  %then we will reset the counters to zero
  TrivialImproveMove = max (0,MaxTrivial - 20);
  DisImproveMove = max (0,MaxDisImp - 20);
 end
```

```
  %if I found a feasible solution and it is better than the current feasible
  %solution
  if feasible==1 && TempCost < ZbestFeas
    if TempCost < Zbest
      Zbest = TempCost;
    end
    XbestFeas = TempXcurrent;
    ZbestFeas = TempCost;
    AvailAcftFeas = TempAvailAcft;
    foundfeas = 1;

    %ELSEIf I found a marginally infeasible solution and it is better than current
    %marginally infeasible solution
  elseif feasible == 2 && TempCost < ZbestMargInfeas
    XbestMargInfeas = TempXcurrent;
    ZbestMargInfeas = TempCost;
    AvailAcftMargInfeas = TempAvailAcft;

    %ELSE If I found a moderately infeasible solution and it is better than current
    %moderately infeasible solution
  elseif feasible==3 && TempCost < ZbestModInfeas
    XbestModInfeas = TempXcurrent;
    ZbestModInfeas = TempCost;
    AvailAcftModInfeas = TempAvailAcft;
  end

  %Best solution out of neghborhood becomes current solution
  Zcurrent = TempCost;
  Xcurrent = TempXcurrent;
  AvailAcft = TempAvailAcft;

end %State

%*************************************************************************
%Update all the counters and stats of the search
%*************************************************************************
if Zcurrent < Zbest %if there is an improving solution

  if Zcurrent*1.1 >= Zbest  %Trivial Solution
    TrivialImproveMove = TrivialImproveMove + 1;
    if foundfeas == 0
      DisImproveMove = 0;
    end
  else %significanlty improving move
    TrivialImproveMove = 0;
    DisImproveMove = 0;
  end
  Xbest = Xcurrent; %best solution found so far
  Zbest = Zcurrent;
  AvailAcftBest = AvailAcft;
```

```
  else %disimproving solution
    DisImproveMove = DisImproveMove + 1;
  end

  %Determine if we are stagnating.  If we are, then start doing more
  %Insert Moves (4 inserts to every one swap)  Otherwise, do all swap moves
  if TrivialImproveMove + DisImproveMove >= 10
    InsVsSwap = -4;
  else
    InsVsSwap = 0;
  end


  %Update the number of aircraft to make sure everything is in order
  NumAcft = size (AvailAcft,1);

  %Update statistics
  TArray (iteration) = TrivialImproveMove;
  DArray (iteration) = DisImproveMove;
  ZArray (iteration) = Zbest;
  text = sprintf ('%1s%.3f%1s','Best solution found so far has a cost of ', Zbest, ...
    '.');
  disp (text)
  disp (')

end %while

%We are done with the tabu search
beep
%Now update all the solutions we are passing out of the function to make sure
%the centerline loaded portion match whether or not an item needs to be
%centerline loaded.  Occationally, this flag gets messed up even though the
%solution is valid
if XbestFeas (1,1,1,1) ~= 0 %if we found a feasible solution...
  [XbestFeas] =FullUpdateXcurrent (XbestFeas,AvailAcftFeas,Cargo,Zones);
end
if size (XbestMargInfeas,1) > 1 %if we found a feasible solution...
  [XbestMargInfeas] =FullUpdateXcurrent (XbestMargInfeas,AvailAcftMargInfeas,Cargo,Zones);
end
if size (XbestModInfeas,1) > 1 %if we found a feasible solution...
  [XbestModInfeas] =FullUpdateXcurrent (XbestModInfeas,AvailAcftModInfeas,Cargo,Zones);
end


%Lastly, compile stats of the feasible solution
if XbestFeas (1,1,1,1) ~= 0 %if we found a feasible solution...
  NumAcft = size (XbestFeas,1);
  t = feastime;
  tottime = cputime - starttime;
  stats = zeros (NumAcft,11);
  for acft = 1:NumAcft
    [LeftCB,RightCB,CB,TargetCB,MinCB,MaxCB,CB_OK,LeftTotalWt,RightTotalWt] = ComputeCB...
```

```
    (acft,AvailAcftFeas,XbestFeas,Cargo,CBLookup);
   [AcftUsageFee,UnderWeightFee,OverWeightFee,CBFee,CBTargetFee,...
    ZoneFee,RampFee]=ComputeCost(AvailAcftFeas,XbestFeas,Cargo,CBLookup,...
    Zones,PalletTable,Costs,acft);
   TotWt = LeftTotalWt+RightTotalWt;
   stats (acft,:) = [TotWt CB MinCB MaxCB TargetCB UnderWeightFee OverWeightFee CBFee ...
    CBTargetFee ZoneFee RampFee];
  end
  XCmatrix = ConvertXcurrent (XbestFeas);
  %if we do not find a feasible solution, then indicate that and then still
  %output the solution to the excel file to help determine why the solution is
  %not feasible. This requires "tricking" the algorithm into thinking it found a
  %feasible solution...thus it outputs a "feasible" solution which is not really
  %feasible...be careful to make sure we don't interpret this as a feasible
  %solution!!!
else
  t = cputime - starttime;
  disp ('Could not find a feasible solution');
  XbestFeas = Xbest;
  NumAcft = size (XbestFeas,1);
  AvailAcftFeas = AvailAcftBest;
  stats = zeros (NumAcft,11);
  for acft = 1:NumAcft
   [LeftCB,RightCB,CB,TargetCB,MinCB,MaxCB,CB_OK,LeftTotalWt,RightTotalWt] = ComputeCB...
    (acft,AvailAcftBest,Xbest,Cargo,CBLookup);
   [AcftUsageFee,UnderWeightFee,OverWeightFee,CBFee,CBTargetFee,...
    ZoneFee,RampFee]=ComputeCost(AvailAcftBest,Xbest,Cargo,CBLookup,...
    Zones,PalletTable,Costs,acft);
   TotWt = LeftTotalWt+RightTotalWt;
   stats (acft,:) = [TotWt CB MinCB MaxCB TargetCB UnderWeightFee OverWeightFee CBFee ...
    CBTargetFee ZoneFee RampFee];
  end
  XCmatrix = ConvertXcurrent (Xbest);
end

%Plot objective function, trivial solutions and disimproving solutions

figure
plot (ZArray);
title ('Objective Function');

figure
subplot (2,1,1);
plot (TArray);
axis ([0 iteration 0 MaxTrivial]);
title ('Trivial Solutions');

subplot (2,1,2);
plot (DArray);
axis ([0 iteration 0 MaxDisImp]);
title ('DisImproving Solutions');
```

```
  % write to the excel file ... this file has a macro which generates a picture
  % of the solution
  disp ('Writing to Excel File')
  xlswrite('Solution Representation.xlsm', XCmatrix, 'Data', 'O2');
  xlswrite('Solution Representation.xlsm', AvailAcftFeas(:,1:13), 'Data', 'A2');
  xlswrite('Solution Representation.xlsm', Cargo, 'Data', 'U2');
  xlswrite('Solution Representation.xlsm', stats, 'Data', 'AO2');
  xlswrite('Solution Representation.xlsm', size (AvailAcftFeas,1), 'Data', 'AZ2');
  system ('Solution Representation.xlsm');
else %Assigns values to output vars if not enough acft to gen init soln
  ZbestFeas = 0;
  ZbestMargInfeas =0;
  ZbestModInfeas = 0;
  XbestFeas = Xcurrent;
  XbestMargInfeas = 0;
  XbestModInfeas = 0;
  AvailAcftFeas = 0;
  AvailAcftMargInfeas = 0;
  AvailAcftModInfeas = 0;
  t = cputime - starttime;
  tottime = cputime-starttime;
end %if success == 1
```

```
function [Xcurrent] = FixLoad (Xcurrent,AcftIndex,AvailAcft,Cargo,...
  CBLookup,Zones,PalletTable,ZoneMult,RampMult)

%*************************************************************************
% Procedure: FixLoad
% Author   : Maj R. Larry Nance
% Purpose  :       Repack the load of an aircraft to try to make it feasible
%                  If there are zone violations present, remove all the rolling
%                  stock from each column.  Replace each item in their respecitve
%                   column in the same order as they were; however, start at the
%                   forward FS and place each item back in a feasible location with
%                   respect to the zones and all the other axle constraints.  Then,
%                   slide the cargo just enough to fix the CB.  Leave the pallets
%                   exactly where they are.  If the load cannot be fixed, then return
%                  the original Xcurrent
% Inputs:
%       Xcurrent            Current solution
%       AcftIndex            Which aircraft are we trying to fix
%       AvailAcft           a x 19 matrix which defines various aspects of the
%                           aircraft availiable for loading (a = number of acft)
%       Cargo:              A n x 20 matrix where n is the number of cargo items.
%                            Columns are defined in the excel file "Pallet Testing"
%       CBLookup             Lookup table for the CB given the weight an acft
%       Zones               Table which defines zone limits and constraints
%       PalletTable         Table which defines pallet locations and constraints
```

```
%        ZoneMult          Zone cost multiplier to determine if there are any zone
%                          violations.
%        RampMult          Ramp cost multiplier to determine if there are any
%                          ramp violations
%
% Outputs:  Xcurrent       New solution that (hopefully) is feasible for
%
% CalledBy: InterAcftInsertRedN
% Calls   : AircraftStats, ComputeZoneCosts, ComputeRampCosts,SlideCBCenter
%*************************************************************************


infeasLoad = 0; %assume the load we have is possible to put on acft
%Gather some statistics on the load --These will not change regarldess of what
%we do within this procedure
[NumLeftRoll,NumLeftPallet,LPallets,NumRtRoll,NumRtPallet,RPallets] =...
  AircraftStats (Xcurrent,AcftIndex,AvailAcft,Cargo,PalletTable);

%Now see if there are any zone violations
[ZCosts,notused1,notused2,notused3,notused4,notused5] = ComputeZoneCosts (AcftIndex,...
  Xcurrent,Cargo,Zones,AvailAcft,PalletTable,ZoneMult);
[RCosts]=ComputeRampCosts(AcftIndex,Xcurrent,AvailAcft,Cargo,...
  NumLeftRoll,NumRtRoll,RampMult);


%Are there any violations and is there any rolling stock? If there are, then fix the load
if (ZCosts > 0 || RCosts > 0) && (NumLeftRoll > 0 || NumRtRoll > 0)


  %Assume NOTHING is centerline loaded...this prevents Pallets loaded in
  %previously centered locations from showing that they are centerline loaded
  Xcurrent(AcftIndex,:,:,4) = 0;

  %Initilaize variables
  TotalRoll = NumLeftRoll + NumRtRoll;
  AcftFSMin = AvailAcft(AcftIndex,3);
  %Figure out where the last FS is on each side where we could load cargo
  if NumLeftPallet > 0
    if max (LPallets(:,3)) == 0
      disp ('debug')
    end
    for i = 1:size(LPallets,1)
      if LPallets(i,3) ~= 0 %if the pallet position is occupied
        LeftAftFS = LPallets(i,1)-1; %One inch forward of most fwd left occupied pallet pos
        break
      end %if
    end%for i = ...
  else
```

```
  LeftAftFS = AvailAcft(AcftIndex,4); %FSMax
end
if NumRtPallet > 0
  if max (RPallets(:,3)) == 0
    disp ('debug')
  end
  found = 0;
  for i = 1:size(RPallets,1)
    if RPallets(i,3) ~= 0 %if the pallet position is occupied
      RtAftFS = RPallets(i,1)-1; %One inch forward of most fwd left occupied pallet pos
      found = 1;
      break
    end
  end%for i = 1:size(RPallet,1)
  if found == 0
    disp ('debug')
  end
else
  RtAftFS = AvailAcft(AcftIndex,4); %FSMax
end

%Gather the axle info for the left side
for i = 1:NumLeftRoll
  iNumAxles = Cargo(Xcurrent (AcftIndex,1,i,3),4);
  if iNumAxles == 0 %If it is a tracked item
    iNumAxles = 1;
  end
  %LeftItems (i,j) = [ItemNumber NumberOfAxles AxleLocation AxleWt]

  for j = 1:iNumAxles
    ItemNumber = Xcurrent (AcftIndex,1,i,3);
    AxleLoc = Cargo(Xcurrent (AcftIndex,1,i,3),4);
    AxleWt = Cargo(Xcurrent (AcftIndex,1,i,3),10+j);
    if iNumAxles == 1
      LeftItems (i,1,1) = ItemNumber;
      LeftItems (i,1,2) = 1;
      LeftItems (i,1,3) = Cargo(Xcurrent (AcftIndex,1,i,3),18); %CB of tracked vehicle
      LeftItems (i,1,4) = Cargo(Xcurrent (AcftIndex,1,i,3),17);
    else
      LeftItems (i,j,1) = ItemNumber;
      LeftItems (i,j,2) = iNumAxles;
      LeftItems (i,j,3) = AxleLoc;
      LeftItems (i,j,4) = AxleWt;
    end %if iNumAxles == 1
  end %for j = 1:iNumAxles
end %for i = 1:NumLeftRoll

%Gather the axle info for the right side
for i = 1:NumRtRoll
```

```
 iNumAxles = Cargo(Xcurrent (AcftIndex,2,i,3),4);
 if iNumAxles == 0 %If it is a tracked item
  iNumAxles = 1;
 end
 %LeftItems (i,j) = [ItemNumber NumberOfAxles AxleLocation AxleWt]

 for j = 1:iNumAxles
  ItemNumber = Xcurrent (AcftIndex,2,i,3);
  AxleLoc = Cargo(Xcurrent (AcftIndex,2,i,3),4);
  AxleWt = Cargo(Xcurrent (AcftIndex,2,i,3),10+j);
  if iNumAxles == 1
   RtItems (i,1,1) = ItemNumber;
   RtItems (i,1,2) = 1;
   RtItems (i,1,3) = Cargo(Xcurrent (AcftIndex,2,i,3),18); %CB of tracked vehicle
   RtItems (i,1,4) = Cargo(Xcurrent (AcftIndex,2,i,3),17);
  else
   RtItems (i,j,1) = ItemNumber;
   RtItems (i,j,2) = iNumAxles;
   RtItems (i,j,3) = AxleLoc;
   RtItems (i,j,4) = AxleWt;
  end %if iNumAxles == 1
 end %for j = 1:iNumAxles
end %for i = 1:NumRtRoll

%*****************************************************************************

%Now, reload Xcurrent
col = 1; %Start with left column
LoadedLeft = zeros (NumLeftRoll,4);
LoadedRt = zeros (NumRtRoll,4);
NumLoadedLeft = 0;
NumLoadedRt = 0;
FSArray = ([AcftFSMin AcftFSMin]);
NumRamps = AvailAcft(AcftIndex,14);
AftRampFS = AvailAcft(AcftIndex,15); %starting FS of aft ramp
if NumRamps == 2
 FwdRampFS = AvailAcft(AcftIndex,18);

end

for i = 1:TotalRoll
 %Which column are we loading into?
 if col == 1
  if NumLoadedLeft+1 <= NumLeftRoll %Load left if there is stuff left to load
   InsertFS = FSArray (1);
  else
   col = 2; %nothing left to load in left side
   InsertFS = FSArray (2);
```

```
  end
else %col == 2
  if NumLoadedRt+1 <= NumRtRoll
    InsertFS = FSArray (2);
  else
    col = 1; %nothing left to load in right side
    InsertFS = FSArray (1);
  end
end

%Now we know where the next insert point is and which column we are loading.
% Now we need to adjust InsertFS back such that we can feasibly load the
% item.



%Insert the item and then slide it aft to fix ramp, axle and zone violations
if col == 1
  NumLoadedLeft = NumLoadedLeft + 1;
  %Gather some information about the cargo item
  %LeftItems (i,j) = [ItemNumber NumberOfAxles AxleLocation AxleWt]
  CargoItem = LeftItems(NumLoadedLeft,1,1);
  NumAxles = LeftItems(NumLoadedLeft,1,2);
  AxleArray = zeros (NumAxles,4);
  CargoWt = Cargo(CargoItem,17);
  CargoWid = Cargo(CargoItem,2);
  CargoLen = Cargo(CargoItem,1);
  AcftType = AvailAcft(AcftIndex,1);
  if Cargo(CargoItem,4) == 0 %tracked
    tracked = 1;
  else
    tracked = 0;
  end
  NumZones = AvailAcft(AcftIndex,7);
  %Load the Item at InsertFS
  LoadedLeft(NumLoadedLeft,1) = InsertFS;
  LoadedLeft(NumLoadedLeft,2) = InsertFS + CargoLen;
  LoadedLeft(NumLoadedLeft,3) = CargoItem;
  for k = 1:NumAxles
    %AxleArray = [FS Weight InZone TooHeavy?]

    if tracked == 1
      AxleArray(k,2) = 0; %don't count the axle wt for the item of tracked vehicle
      AxleArray(k,1) = InsertFS + Cargo(CargoItem,18); %The CB of the item
    else
      AxleArray(k,1) = InsertFS + Cargo(CargoItem,k+ 4);
      AxleArray(k,2) = Cargo(CargoItem,k + 10);
    end
```

```
 for j = 1:NumZones
  %if the FS of the axle is in the kth zone
  if AxleArray(k,1) >= Zones(AcftType,j,1,2) &&...
     AxleArray(k,1) <= Zones(AcftType,j,1,3)
    AxleArray(k,3) = j;
    if AxleArray(k,2) > Zones(AcftType,j,1,4)
     AxleArray(k,4) = 1;
    else
     AxleArray(k,4) = 0;
    end
    break
  end

 end %for j
end %for k



%*********************************
%Step 1:  Find Zone in which the axle
%weights and  will fit.  Needs to be as close to
%InsertFS as possible

%Are there any axles that are too heavy for their zones.  If YES, then find
%the nearest zone that will accomodate an axle of that weight
%Keep looping through this until all the axles are good
found = 1;
iteration = 1;
while found == 1 && iteration < 7
 found = 0;
 [x index] = find (AxleArray(:,4) == 1);
 if ~isempty(x) %if there is an axle that has a zone violation
  %If it finds more than one item that is overweight, pick the last one
  if size (x,1) > 1
   x = x(size(x,1));
  end
  %Figure out nearest zone that can accomodate an axle of that weight
  found = 1;
  InZone = AxleArray(x,3);
  Diff = inf; %want to pick closest zone to curent zone
  WantZone = 0;
  for zonecount = InZone:NumZones

   if AxleArray(x,2) < Zones(AcftType,zonecount,1,4) &&...
      abs(zonecount-InZone) < Diff
     Diff = abs(zonecount-InZone);
     WantZone = zonecount;
   end
```

```
    end %for
    if WantZone ~= 0 %if there is a zone I can slide this to...

      ZoneFS = Zones(AcftType,WantZone,1,2);
      delta = ZoneFS - AxleArray(x,1);
    else
      delta = -1; %force the program to indicate infeasible load
    end
    %Now slide the cargo and axles aft by Diff
    if delta > 0 %can only go to a farther aft zone.
      InsertFS = InsertFS + delta+1;
      LoadedLeft(NumLoadedLeft,1) = InsertFS;
      LoadedLeft(NumLoadedLeft,2) = InsertFS + CargoLen;
      for s = 1:NumAxles
        %AxleArray = [FS Weight InZone TooHeavy?]
        if tracked == 1
          AxleArray(s,2) = 0; %don't count the axle wt for the item of tracked vehicle
          AxleArray(s,1) = InsertFS + Cargo(CargoItem,18); %The CB of the item
        else
          AxleArray(s,1) = InsertFS + Cargo(CargoItem,s+ 4);
          AxleArray(s,2) = Cargo(CargoItem,s + 10);
        end
        for p = 1:NumZones
          %if the FS of the axle is in the kth zone
          if AxleArray(s,1) >= Zones(AcftType,p,1,2) &&...
              AxleArray(s,1) <= Zones(AcftType,p,1,3)
            AxleArray(s,3) = p;
            if AxleArray(s,2) > Zones(AcftType,p,1,4)
              AxleArray(s,4) = 1;
            else
              AxleArray(s,4) = 0;
            end
            break
          end
          %check to see if it is too heavy

        end %for p
      end %for s
    else
      found = 0; %force exit the while loop b/c can't load this load
      infeasLoad = 1;
    end
  end %if ~isempty
  iteration = iteration + 1;
end%while

if infeasLoad == 1
  break
end
```

```
%Now, all the axles on the cargo item should be in zones that can support
%them

%*************************************************************************
%Step 2:

%Now check to see if we are on the forward ramp
if NumRamps  == 2 && InsertFS < FwdRampFS %if any part of item hangs over ramp
 %Rules:  1) Tracked items cannot span across the ramp
 %       2) Items must have one wheel on the ramp and one off the ramp if
 %       they span the ramp
 %       3) Items cannot overhang ramp if their wheels are not on it.

 %Rule 1)  Tracked
 if (tracked == 1) && (InsertFS + CargoLen > FwdRampFS) %if tracked and spans ramp
  %then move the item aft off of the ramp
  delta = FwdRampFS - InsertFS; %pos #
  InsertFS = InsertFS + delta;
  LoadedLeft(NumLoadedLeft,1) = InsertFS ;
  LoadedLeft(NumLoadedLeft,2) = InsertFS + CargoLen;
  for k = 1:NumAxles
   %NOTE: AxleArray = [FS Weight]
   if tracked == 1
    AxleArray(k,2) = 0; %don't count the axle wt for the item of tracked vehicle
    AxleArray(k,1) = InsertFS + Cargo(CargoItem,18); %The CB of the item
   else
    AxleArray(k,1) = InsertFS + Cargo(CargoItem,k+ 4);
    AxleArray(k,2) = Cargo(CargoItem,k + 10);
   end
  end %for k

 end %if tracked == 1


 %Rule 2)  If Items span the ramp, they must have an axle on the front ramp
 %and one on the aft
 if InsertFS+12 < FwdRampFS && InsertFS + CargoLen > FwdRampFS
  FrontAxleFS = AxleArray(1,1);
  AftAxleFS = AxleArray(NumAxles,1);
  if FrontAxleFS > FwdRampFS || AftAxleFS < FwdRampFS %if violation
   %shift aft by the distance from the Fwd Ramp front of the item
   delta = FwdRampFS-(InsertFS+12); %pos #
   InsertFS = InsertFS + delta;
   LoadedLeft(NumLoadedLeft,1) = InsertFS ;
   LoadedLeft(NumLoadedLeft,2) = InsertFS + CargoLen;
   for k = 1:NumAxles
    %NOTE: AxleArray = [FS Weight]
```

```
      if tracked == 1
        AxleArray(k,2) = 0; %don't count the axle wt for the item of tracked vehicle
        AxleArray(k,1) = InsertFS + Cargo(CargoItem,18); %The CB of the item
      else
        AxleArray(k,1) = InsertFS + Cargo(CargoItem,k+ 4);
        AxleArray(k,2) = Cargo(CargoItem,k + 10);
      end
    end %for k
  end %if FrontAxleFS > ...
end %if InsertFS < FwdRampFS

 %Rule 3)  Items cannot overhang ramp if their wheels are not the ramp
 if InsertFS +13  < FwdRampFS && AxleArray(1,1) > FwdRampFS
   delta = FwdRampFS - (InsertFS+12);
   InsertFS = InsertFS + delta;
   LoadedLeft(NumLoadedLeft,1) = InsertFS ;
   LoadedLeft(NumLoadedLeft,2) = InsertFS + CargoLen;
   for k = 1:NumAxles
     %NOTE: AxleArray = [FS Weight]
     if tracked == 1
       AxleArray(k,2) = 0; %don't count the axle wt for the item of tracked vehicle
       AxleArray(k,1) = InsertFS + Cargo(CargoItem,18); %The CB of the item
     else
       AxleArray(k,1) = InsertFS + Cargo(CargoItem,k+ 4);
       AxleArray(k,2) = Cargo(CargoItem,k + 10);
     end
   end %for k
 end %if InsertFS + 12
end %if NumRamps == 2
%if we have exceeded the aft of the cargo compartment then quit
if LoadedLeft (NumLoadedLeft,2) > LeftAftFS
  infeasLoad = 1;
  break
end
%Now update the zones the Axles are in
for s = 1:NumAxles
  for p = 1:NumZones
    %if the FS of the axle is in the kth zone
    if AxleArray(s,1) >= Zones(AcftType,p,1,2) &&...
       AxleArray(s,1) <= Zones(AcftType,p,1,3)
      AxleArray(s,3) = p;
      break
    end
  end %for p
end %for s

%*************************************************************************
%Now check to see if the item needs to be centerline loaded based on its
%weight and width
```

```
for index = 1:NumAxles
  InZone = AxleArray(index,3);  %Gather which zone we are looking at
  if CargoWt > Zones(AcftType,InZone,1,10) ||... %if should be centerline loaded (weight)
     CargoWid > AvailAcft(AcftIndex,5) %if it should be centerline loaded (width)
    %The item needs to be centerline loaded, so shift the opposite column's
    %FS pointer to the end of this cargo item
    %We need to make sure not to centerline load any cargo AFT of a
    %pallet.  So, if there is a pallet on the right side, we need to make
    %sure we don't try to centerline load aft of that pallet position
    %(Denoted by RtAftFS

    Center = 1;
    if FSArray(2) > InsertFS %need to be past the last right item
      InsertFS = FSArray(2);
    end
    if InsertFS + CargoLen < RtAftFS
      LoadedLeft(NumLoadedLeft,1) = InsertFS ;
      LoadedLeft(NumLoadedLeft,2) = InsertFS + CargoLen;

      for k = 1:NumAxles
        %NOTE: AxleArray = [FS Weight]
        if tracked == 1
          AxleArray(k,2) = 0; %don't count the axle wt for the item of tracked vehicle
          AxleArray(k,1) = InsertFS + Cargo(CargoItem,18); %The CB of the item
        else
          AxleArray(k,1) = InsertFS + Cargo(CargoItem,k+ 4);
          AxleArray(k,2) = Cargo(CargoItem,k + 10);
        end
      end %for k
      break %Found the item...so quit
    else
      infeasLoad = 1;  %trying to insert a centered item aft of rt side pallets
    end %if InsertFS...
  else
    Center = 0;
  end%if
end %for index
if infeasLoad == 1
  break %quit if the load is infeasible
end
if Center == 1
  FSArray(2) = LoadedLeft(NumLoadedLeft,2) + 1; %Right side index incremented
  LoadedLeft(NumLoadedLeft,4) = 1; %Indicate Centered
end

%*************************************************************************
%Now check for axles that are too heavy next to each other only if item is
```

```
%not centerline loaded
if Center == 0 && tracked == 0
 FSFwd = LoadedLeft(NumLoadedLeft,1);
 FSAft = LoadedLeft(NumLoadedLeft,2);

 %Determine SubZones for each of the Axles in the Left Side
 for k = 1:NumAxles
  InZone = AxleArray(k,3);

  for subzone = 1:Zones(AcftType,InZone,1) %# of possible subzones
   weight = AxleArray(k,2);

   if weight >= Zones(AcftType,InZone,subzone,5) &&...
      weight <= Zones(AcftType,InZone,subzone,6) %if its weight is in the subzone
    SubZone(k) = subzone;
    break
   end
  end %for subzone
 end %for k

 %Now, need to look through each item in the right side to see if it
 %overlaps the left column item we are inserting
 for rtcol = 1:NumLoadedRt
  if Cargo(LoadedRt(rtcol,3),4) == 0
   RtTracked =1;
  else
   RtTracked = 0;
  end
  if RtTracked == 0
   RightFSFwd = LoadedRt(rtcol,1);
   RightFSAft = LoadedRt(rtcol,2);

   %If the item being inserted is next to an item in the right column...
   if RightFSFwd >= FSFwd && RightFSFwd <= FSAft || ...
      RightFSAft >= FSFwd && RightFSFwd <= FSAft || ...
      RightFSFwd >= FSAft && RightFSFwd <= FSAft || ...
      RightFSAft >= FSAft && RightFSFwd <= FSAft
    %Need to check all the axles of the right item
    NumRtAxles = Cargo(LoadedRt(rtcol,3),4);

    for rtAxles = 1:NumRtAxles
     for leftAxles = 1:NumAxles
      leftFS = InsertFS + Cargo(CargoItem,leftAxles + 4);
      leftWt = Cargo(CargoItem,leftAxles + 10);
      rtFS = RightFSFwd + Cargo(LoadedRt(rtcol,3),rtAxles+4);
      rtWt = Cargo(LoadedRt(rtcol,3),rtAxles+10);
      InZone = AxleArray(leftAxles,3);
      InSubZone = SubZone (leftAxles);
```

```
        ReqSep = Zones(AcftType,InZone,1,9);
        if abs(leftFS - rtFS) <= ReqSep  %if they are close to one another

          Slope = Zones(AcftType,InZone,InSubZone,7);
          Intercept =Zones(AcftType,InZone,InSubZone,8);
          MaxRtWt = leftWt * Slope + Intercept;
          if MaxRtWt < rtWt %if they are too heavy then slide the left item down
            delta = ReqSep +1 - abs(leftFS - rtFS);
            InsertFS = InsertFS + delta;
            LoadedLeft(NumLoadedLeft,1) = InsertFS ;
            LoadedLeft(NumLoadedLeft,2) = InsertFS + CargoLen;
            for k = 1:NumAxles
              %NOTE: AxleArray = [FS Weight]
              if tracked == 1
                AxleArray(k,2) = 0; %don't count the axle wt for the item of tracked vehicle
                AxleArray(k,1) = InsertFS + Cargo(CargoItem,18); %The CB of the item
              else
                AxleArray(k,1) = InsertFS + Cargo(CargoItem,k+ 4);
                AxleArray(k,2) = Cargo(CargoItem,k + 10);
              end
            end %for k
          end %MaxRtWt > rtWt
        end %  if abs(leftFS - rtFS) <= ReqSe



      end %for leftAxles = 1:NumAxles

    end %rtAxles = 1:NumRtAxles
  end %RightFSFwd >= FSFwd && RightFSFwd <= FSAft || ...
end %if RtTracked = 0

end %for rtcol = ...
%Update LoadedLeft/Right and Axles only if InsertFS changed




end %if Center == 0 && Tracked == 0

%*************************************************************************
%Lastly, need to make sure the aft ramp is configured correclty
%Rules:  1) Tracked items cannot span across the ramp
%       2) Items must have one wheel on the ramp and one off the ramp if
%       they span the ramp
%       3) Items cannot overhang ramp if their wheels are not on it.
```

```
%Rule 1)  Tracked
if (tracked == 1) && (InsertFS + CargoLen > AftRampFS) && ...
   (InsertFS < AftRampFS) %if tracked and spans ramp
  %then move the item aft off of the ramp
  delta = AftRampFS - InsertFS; %pos #
  InsertFS = InsertFS + delta;
  LoadedLeft(NumLoadedLeft,1) = InsertFS ;
  LoadedLeft(NumLoadedLeft,2) = InsertFS + CargoLen;
  for k = 1:NumAxles
    %NOTE: AxleArray = [FS Weight]
    if tracked == 1
      AxleArray(k,2) = 0; %don't count the axle wt for the item of tracked vehicle
      AxleArray(k,1) = InsertFS + Cargo(CargoItem,18); %The CB of the item
    else
      AxleArray(k,1) = InsertFS + Cargo(CargoItem,k+ 4);
      AxleArray(k,2) = Cargo(CargoItem,k + 10);
    end
  end %for k
  InsertFS = InsertFS + delta;
end %if tracked == 1


%Rule 2)  If Items span the ramp, they must have an axle on the aft ramp
%and one on the aft
if InsertFS < AftRampFS && InsertFS + CargoLen > AftRampFS
  FrontAxleFS = AxleArray(1,1);
  AftAxleFS = AxleArray(NumAxles,1);
  if FrontAxleFS > AftRampFS || AftAxleFS < AftRampFS %if violation
    %shift aft by the distance from the Fwd Ramp to the Aft Axle
    if AftAxleFS < AftRampFS
      delta = AftRampFS - AftAxleFS+1; %pos #
    else
      delta = AftRampFS - InsertFS;
    end
    if delta < 0
      disp ('debug')
    end
    InsertFS = InsertFS + delta;
    LoadedLeft(NumLoadedLeft,1) = InsertFS ;
    LoadedLeft(NumLoadedLeft,2) = InsertFS + CargoLen;
    for k = 1:NumAxles
      %NOTE: AxleArray = [FS Weight]
      if tracked == 1
        AxleArray(k,2) = 0; %don't count the axle wt for the item of tracked vehicle
        AxleArray(k,1) = InsertFS + Cargo(CargoItem,18); %The CB of the item
      else
        AxleArray(k,1) = InsertFS + Cargo(CargoItem,k+ 4);
        AxleArray(k,2) = Cargo(CargoItem,k + 10);
      end
```

```
    end %for k
   end %if FrontAxleFS > ...
 end %if InsertFS < FwdRampFS

 %Rule 3)  Items cannot overhang ramp if their wheels are not the ramp
 if LoadedLeft(NumLoadedLeft,2) - 12  > AftRampFS && AxleArray(NumAxles,1) < AftRampFS
   delta = AftRampFS - (LoadedLeft(NumLoadedLeft,2) - 12);
   InsertFS = InsertFS + delta;
   LoadedLeft(NumLoadedLeft,1) = InsertFS ;
   LoadedLeft(NumLoadedLeft,2) = InsertFS + CargoLen;
   for k = 1:NumAxles
    %NOTE: AxleArray = [FS Weight]
    AxleArray(k,1) = InsertFS + Cargo(CargoItem,4+k);
    if tracked == 1
     AxleArray(k,2) = 0; %don't count the axle wt for the item of tracked vehicle
    else
     AxleArray(k,2) = Cargo(CargoItem,k + 10);
    end
   end %for k
 end %if InsertFS + 12

 %UPDATE Xcurrent
 FSArray(1) = LoadedLeft(NumLoadedLeft,2) + 1;
 if FSArray(1) > LeftAftFS
   infeasLoad = 1;
   break;
 end
 col = 2; %switch to the other column
 %*************************************************************************
 %*******LOADING ON RIGHT %SIDE*******************************************
 %*************************************************************************

else %col == 2
 %Insert the item and then slide it aft to fix ramp, axle and zone violations
 NumLoadedRt = NumLoadedRt + 1;
 %Gather some information about the cargo item
 %LeftItems (i,j) = [ItemNumber NumberOfAxles AxleLocation AxleWt]
 CargoItem = RtItems(NumLoadedRt,1,1);
 NumAxles = RtItems(NumLoadedRt,1,2);
 AxleArray = zeros (NumAxles,4);
 CargoWt = Cargo(CargoItem,17);
 CargoWid = Cargo(CargoItem,2);
 CargoLen = Cargo(CargoItem,1);
 AcftType = AvailAcft(AcftIndex,1);
 if Cargo(CargoItem,4) == 0 %tracked
   tracked = 1;
 else
   tracked = 0;
 end
```

```
NumZones = AvailAcft(AcftIndex,7);
%Load the Item at InsertFS
LoadedRt(NumLoadedRt,1) = InsertFS;
LoadedRt(NumLoadedRt,2) = InsertFS + CargoLen;
LoadedRt(NumLoadedRt,3) = CargoItem;
for k = 1:NumAxles
 %AxleArray = [FS Weight InZone TooHeavy?]

 if tracked == 1
  AxleArray(k,2) = 0; %don't count the axle wt for the item of tracked vehicle
  AxleArray(k,1) = InsertFS + Cargo(CargoItem,18); %The CB of the item
 else
  AxleArray(k,1) = InsertFS + Cargo(CargoItem,k+ 4);
  AxleArray(k,2) = Cargo(CargoItem,k + 10);
 end
 for j = 1:NumZones
  %if the FS of the axle is in the kth zone
  if AxleArray(k,1) >= Zones(AcftType,j,1,2) &&...
     AxleArray(k,1) <= Zones(AcftType,j,1,3)
    AxleArray(k,3) = j;
    if AxleArray(k,2) > Zones(AcftType,j,1,4)
     AxleArray(k,4) = 1;
    else
     AxleArray(k,4) = 0;
    end
    break
  end

 end %for j
end %for k



%*********************************
%Step 1:  Find Zone in which the axle
%weights and  will fit.  Needs to be as close to
%InsertFS as possible

%Are there any axles that are too heavy for their zones.  If YES, then find
%the nearest zone that will accomodate an axle of that weight
%Keep looping through this until all the axles are good
found = 1;
iteration = 1;
while found == 1 && iteration < 10
 found = 0;
 [x index] = find (AxleArray(:,4) == 1);
 if ~isempty(x) %if there is an axle that has a zone violation
  %Figure out nearest zone that can accomodate an axle of that weight
```

```
found = 1;
%If it finds more than one item that is overweight, pick the last one
if size (x,1) > 1
  x = x(size(x,1));
end
inZone = AxleArray(x,3);
Diff = inf; %want to pick closest zone to curent zone
WantZone = 0;
for zonecount = inZone:NumZones

  if AxleArray(x,2) < Zones(AcftType,zonecount,1,4) &&...
     abs(zonecount-inZone) < Diff
    Diff = abs(zonecount-inZone);
    WantZone = zonecount;
  end
end %for
if WantZone ~= 0 %if there is a zone I can slide this to...

  ZoneFS = Zones(AcftType,WantZone,1,2);
  delta = ZoneFS - AxleArray(x,1);
else
  delta = -1; %force the program to indicate infeasible load
end
%Now slide the cargo and axles aft by Diff
if delta > 0
  InsertFS = InsertFS + delta+1;
  LoadedRt(NumLoadedRt,1) = InsertFS;
  LoadedRt(NumLoadedRt,2) = InsertFS + CargoLen;
  for s = 1:NumAxles
    %AxleArray = [FS Weight InZone TooHeavy?]
    if tracked == 1
      AxleArray(s,2) = 0; %don't count the axle wt for the item of tracked vehicle
      AxleArray(s,1) = InsertFS + Cargo(CargoItem,18); %The CB of the item
    else
      AxleArray(s,1) = InsertFS + Cargo(CargoItem,s+ 4);
      AxleArray(s,2) = Cargo(CargoItem,s + 10);
    end
    for p = 1:NumZones
      %if the FS of the axle is in the kth zone
      if AxleArray(s,1) >= Zones(AcftType,p,1,2) &&...
         AxleArray(s,1) <= Zones(AcftType,p,1,3)
        AxleArray(s,3) = p;
        if AxleArray(s,2) > Zones(AcftType,p,1,4)
          AxleArray(s,4) = 1;
        else
          AxleArray(s,4) = 0;
        end
        break
      end
```

```
        %check to see if it is too heavy

       end %for p
      end %for s
    else %delta < 0, so we have hit an infeasible load
     infeasLoad = 1;
     found = 0; %force the loop to exit
    end %if delta > 0
  end %if ~isempty
  iteration = iteration + 1;
end%while
if iteration == 10
  disp ('debug')
end
if infeasLoad == 1
  break %stop trying to load the acft
end




%Now, all the axles on the cargo item should be in zones that can support
%them

%*************************************************************************
%Step 2:

%Now check to see if we are on the forward ramp
if NumRamps  == 2 && InsertFS < FwdRampFS %if any part of item hangs over ramp
  %Rules:  1) Tracked items cannot span across the ramp
  %       2) Items must have one wheel on the ramp and one off the ramp if
  %       they span the ramp
  %       3) Items cannot overhang ramp if their wheels are not on it.

  %Rule 1)  Tracked
  if (tracked == 1) && (InsertFS + CargoLen > FwdRampFS) %if tracked and spans ramp
    %then move the item aft off of the ramp
    delta = FwdRampFS - InsertFS; %pos #
    InsertFS = InsertFS + delta;
    LoadedRt(NumLoadedRt,1) = InsertFS ;
    LoadedRt(NumLoadedRt,2) = InsertFS + CargoLen;
    for k = 1:NumAxles
      %NOTE: AxleArray = [FS Weight]
      if tracked == 1
        AxleArray(k,2) = 0; %don't count the axle wt for the item of tracked vehicle
        AxleArray(k,1) = InsertFS + Cargo(CargoItem,18); %The CB of the item
      else
        AxleArray(k,1) = InsertFS + Cargo(CargoItem,k+ 4);
```

```
    AxleArray(k,2) = Cargo(CargoItem,k + 10);
   end
  end %for k

end %if tracked == 1


%Rule 2)  If Items span the ramp, they must have an axle on the front ramp
%and one on the aft
if InsertFS+12 < FwdRampFS && InsertFS + CargoLen > FwdRampFS
  FrontAxleFS = AxleArray(1,1);
  AftAxleFS = AxleArray(NumAxles,1);
  if FrontAxleFS > FwdRampFS || AftAxleFS < FwdRampFS %if violation
   %shift aft by the distance from the Front of vechicle to the Ramp
   delta = FwdRampFS - (InsertFS +13); %pos #
   InsertFS = InsertFS + delta;
   LoadedRt(NumLoadedRt,1) = InsertFS ;
   LoadedRt(NumLoadedRt,2) = InsertFS + CargoLen;
   for k = 1:NumAxles
    %NOTE: AxleArray = [FS Weight]

    if tracked == 1
     AxleArray(k,2) = 0; %don't count the axle wt for the item of tracked vehicle
     AxleArray(k,1) = InsertFS + Cargo(CargoItem,18); %The CB of the item
    else
     AxleArray(k,1) = InsertFS + Cargo(CargoItem,k+ 4);
     AxleArray(k,2) = Cargo(CargoItem,k + 10);
    end
   end %for k
  end %if FrontAxleFS > ...
end %if InsertFS < FwdRampFS

%Rule 3)  Items cannot overhang ramp if their wheels are not the ramp
if InsertFS +13  < FwdRampFS && AxleArray(1,1) > FwdRampFS
  delta = FwdRampFS - (InsertFS+13);
  InsertFS = InsertFS + delta;
  LoadedRt(NumLoadedRt,1) = InsertFS ;
  LoadedRt(NumLoadedRt,2) = InsertFS + CargoLen;
  for k = 1:NumAxles
   %NOTE: AxleArray = [FS Weight]
   if tracked == 1
    AxleArray(k,2) = 0; %don't count the axle wt for the item of tracked vehicle
    AxleArray(k,1) = InsertFS + Cargo(CargoItem,18); %The CB of the item
   else
    AxleArray(k,1) = InsertFS + Cargo(CargoItem,k+ 4);
    AxleArray(k,2) = Cargo(CargoItem,k + 10);
   end
  end %for k
```

```
  end %if InsertFS + 12
end %if NumRamps == 2

%if we have exceeded the aft of the cargo compartment then quit
if LoadedRt (NumLoadedRt,2) > RtAftFS
  infeasLoad = 1;
  break
end
%Now update the zones the Axles are in
for s = 1:NumAxles
  for p = 1:NumZones
    %if the FS of the axle is in the kth zone
    if AxleArray(s,1) >= Zones(AcftType,p,1,2) &&...
       AxleArray(s,1) <= Zones(AcftType,p,1,3)
      AxleArray(s,3) = p;
      break
    end
  end %for p
end %for s

%*************************************************************************
%Now check to see if the item needs to be centerline loaded based on its
%weight and width
for index = 1:NumAxles %If the axle has no zone, it is off end of cargo compartment

  InZone = AxleArray(index,3);  %Gather which zone we are looking at
  if CargoWt > Zones(AcftType,InZone,1,10) ||... %if should be centerline loaded (weight)
     CargoWid > AvailAcft(AcftIndex,5) %if it should be centerline loaded (width)
    %The item needs to be centerline loaded, so shift the opposite column's
    %FS pointer to the end of this cargo item
    Center = 1;
    if FSArray(1) > InsertFS %need to be past the last right item
      InsertFS = FSArray(1) ;
    end
    if InsertFS +CargoLen < LeftAftFS %if no blocking pallets on left side
      LoadedRt(NumLoadedRt,1) = InsertFS ;
      LoadedRt(NumLoadedRt,2) = InsertFS + CargoLen;

      for k = 1:NumAxles
        %NOTE: AxleArray = [FS Weight InZone TooHeavy?]
        if tracked == 1
          AxleArray(k,2) = 0; %don't count the axle wt for the item of tracked vehicle
          AxleArray(k,1) = InsertFS + Cargo(CargoItem,18); %The CB of the item
          for p = 1:NumZones
            %if the FS of the axle is in the kth zone
            if AxleArray(k,1) >= Zones(AcftType,p,1,2) &&...
               AxleArray(k,1) <= Zones(AcftType,p,1,3)
              AxleArray(k,3) = p;
```

```
          break
        end
      end %for p
    else
      AxleArray(k,1) = InsertFS + Cargo(CargoItem,k+ 4);
      AxleArray(k,2) = Cargo(CargoItem,k + 10);
      for p = 1:NumZones
        %if the FS of the axle is in the kth zone
        if AxleArray(k,1) >= Zones(AcftType,p,1,2) &&...
            AxleArray(k,1) <= Zones(AcftType,p,1,3)
          AxleArray(k,3) = p;
          break
        end %if AxleArray(k,1)...
      end %for p
    end %if tracked ==1
  end %for k


   break
  else %there is a pallet blocking the insert
   infeasLoad = 1;
  end%if InsertFS +CargoLen < LeftAftFS

 else
  Center = 0;
 end%if
end %for index
if Center == 1
 FSArray(1) = LoadedRt(NumLoadedRt,2) + 1; %Left side index incremented
 LoadedRt(NumLoadedRt,4) = 1; %Indicate Centered
 %It is possible that this may return a "centered" item which is not
 %really centered because there are pallets in the way.  This should not
 %be a problem because the solution will have a high zone cost and won't
 %be used as the best solution...
end
if infeasLoad == 1
 break
end

%*************************************************************************
%Now check for axles that are too heavy next to each other only if item is
%not centerline loaded
if Center == 0 && tracked == 0
 FSFwd = LoadedRt(NumLoadedRt,1);
 FSAft = LoadedRt(NumLoadedRt,2);

 %Determine SubZones for each of the Axles in the LRighteft Side
 for k = 1:NumAxles
```

```
 InZone = AxleArray(k,3);
 if InZone == 0
  disp ('debug')
 end
 for subzone = 1:Zones(AcftType,InZone,1) %# of possible subzones
  weight = AxleArray(k,2);

  if weight >= Zones(AcftType,InZone,subzone,5) &&...
     weight <= Zones(AcftType,InZone,subzone,6) %if its weight is in the subzone
   SubZone(k) = subzone;
   break
  end
 end %for subzone
end %for k

%Now, need to look through each item in the right side to see if it
%overlaps the left column item we are inserting
for leftcol = 1:NumLoadedLeft
 if Cargo(LoadedLeft(leftcol,3),4) == 0
  LeftTracked =1;
 else
  LeftTracked = 0;
 end
 if LeftTracked == 0
  LeftFSFwd = LoadedLeft(leftcol,1);
  LeftFSAft = LoadedLeft(leftcol,2);

  %If the item being inserted is next to an item in the right column...
  if LeftFSFwd >= FSFwd && LeftFSFwd <= FSAft || ...
     LeftFSAft >= FSFwd && LeftFSFwd <= FSAft || ...
     LeftFSFwd >= FSAft && LeftFSFwd <= FSAft || ...
     LeftFSAft >= FSAft && LeftFSFwd <= FSAft
   %Need to check all the axles of the left item
   NumLeftAxles = Cargo(LoadedLeft(leftcol,3),4);

   for leftAxles = 1:NumLeftAxles
    for rtAxles = 1:NumAxles

     leftFS = InsertFS + Cargo(LoadedLeft(leftcol,3),leftAxles + 4);
     leftWt = Cargo(LoadedLeft(leftcol,3),leftAxles + 10);
     rtFS = InsertFS + Cargo(CargoItem,rtAxles+4);
     rtWt = Cargo(CargoItem,rtAxles+10);
     InZone = AxleArray(rtAxles,3);
     if InZone == 0
      disp ('debug')
     end
     InSubZone = SubZone (rtAxles);
     if InSubZone == 0
```

```
          disp ('debug')
        end
        ReqSep = Zones(AcftType,InZone,1,9);
        if abs(leftFS - rtFS) <= ReqSep  %if they are close to one another

          Slope = Zones(AcftType,InZone,InSubZone,7);
          Intercept =Zones(AcftType,InZone,InSubZone,8);
          MaxLeftWt = rtWt * Slope + Intercept;
          if MaxLeftWt < leftWt %if they are too heavy then slide the left item down
            delta = ReqSep + 1 - abs(leftFS - rtFS);
            InsertFS = InsertFS + delta;
            %Update LoadedLeft/Right and Axles
            LoadedRt(NumLoadedRt,1) = InsertFS ;
            LoadedRt(NumLoadedRt,2) = InsertFS + CargoLen;
            for k = 1:NumAxles
              %NOTE: AxleArray = [FS Weight]
              if tracked == 1
                AxleArray(k,2) = 0; %don't count the axle wt for the item of tracked vehicle
                AxleArray(k,1) = InsertFS + Cargo(CargoItem,18); %The CB of the item
              else
                AxleArray(k,1) = InsertFS + Cargo(CargoItem,k+ 4);
                AxleArray(k,2) = Cargo(CargoItem,k + 10);
              end
            end %for k
          end %MaxRtWt > rtWt
        end %  if abs(leftFS - rtFS) <= ReqSe



      end %for leftAxles = 1:NumAxles

    end %rtAxles = 1:NumRtAxles
  end %RightFSFwd >= FSFwd && RightFSFwd <= FSAft || ...
 end %if RtTracked = 0

end %for rtcol = ...

end %if Center == 0 && Tracked == 0

%*****************************************************************************
%Lastly, need to make sure the aft ramp is configured correclty
%Rules:  1) Tracked items cannot span across the ramp
%       2) Items must have one wheel on the ramp and one off the ramp if
%       they span the ramp
%       3) Items cannot overhang ramp if their wheels are not on it.

%Rule 1)  Tracked
```

```
if (tracked == 1) && (InsertFS + CargoLen > AftRampFS) && ...
   (InsertFS < AftRampFS) %if tracked and spans ramp
  %then move the item aft onto the ramp of the ramp
  delta = AftRampFS - InsertFS; %pos #
  InsertFS = InsertFS + delta;
  LoadedRt(NumLoadedRt,1) = InsertFS ;
  LoadedRt(NumLoadedRt,2) = InsertFS + CargoLen;
  for k = 1:NumAxles
    %NOTE: AxleArray = [FS Weight]
    if tracked == 1
      AxleArray(k,2) = 0; %don't count the axle wt for the item of tracked vehicle
      AxleArray(k,1) = InsertFS + Cargo(CargoItem,18); %The CB of the item
    else
      AxleArray(k,1) = InsertFS + Cargo(CargoItem,k+ 4);
      AxleArray(k,2) = Cargo(CargoItem,k + 10);
    end
  end %for k
  InsertFS = InsertFS + delta;
end %if tracked == 1


%Rule 2)  If Items span the ramp, they must have an axle on the aft ramp
%and one on the aft
if InsertFS < AftRampFS && InsertFS + CargoLen > AftRampFS
  FrontAxleFS = AxleArray(1,1);
  AftAxleFS = AxleArray(NumAxles,1);
  if FrontAxleFS > AftRampFS || AftAxleFS < AftRampFS %if violation
    %shift aft by the distance from the Fwd Ramp to front of the item
    %(Only option is to slide forward)
    if AftAxleFS < AftRampFS
      delta = AftRampFS - AftAxleFS+1; %pos #
    else
      delta = AftRampFS - InsertFS+1;
    end
    if delta < 0
      disp ('debug')
    end
    InsertFS = InsertFS + delta;
    LoadedRt(NumLoadedRt,1) = InsertFS ;
    LoadedRt(NumLoadedRt,2) = InsertFS + CargoLen;
    for k = 1:NumAxles
      %NOTE: AxleArray = [FS Weight]
      if tracked == 1
        AxleArray(k,2) = 0; %don't count the axle wt for the item of tracked vehicle
        AxleArray(k,1) = InsertFS + Cargo(CargoItem,18); %The CB of the item
      else
        AxleArray(k,1) = InsertFS + Cargo(CargoItem,k+ 4);
        AxleArray(k,2) = Cargo(CargoItem,k + 10);
      end
```

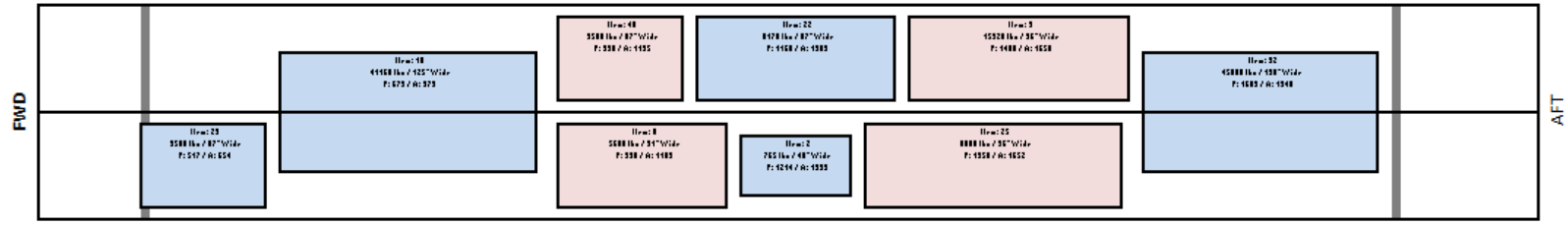
```
      end %for k
    end %if FrontAxleFS > ...
  end %if InsertFS < FwdRampFS

  %Rule 3)  Items cannot overhang ramp if their wheels are not the ramp
  if LoadedRt(NumLoadedRt,2) - 12  > AftRampFS && AxleArray(NumAxles,1) < AftRampFS
    delta = AftRampFS - (LoadedRt(NumLoadedRt,2) - 12);
    InsertFS = InsertFS + delta;
    LoadedRt(NumLoadedRt,1) = InsertFS ;
    LoadedRt(NumLoadedRt,2) = InsertFS + CargoLen;
    for k = 1:NumAxles
     %NOTE: AxleArray = [FS Weight]
     if tracked == 1
       AxleArray(k,2) = 0; %don't count the axle wt for the item of tracked vehicle
       AxleArray(k,1) = InsertFS + Cargo(CargoItem,18); %The CB of the item
     else
       AxleArray(k,1) = InsertFS + Cargo(CargoItem,k+ 4);
       AxleArray(k,2) = Cargo(CargoItem,k + 10);
     end
    end %for k
  end %if InsertFS + 12

  %Update Array
  FSArray(2) = LoadedRt(NumLoadedRt,2) + 1;
  if LoadedRt(NumLoadedRt,2)  > RtAftFS
    infeasLoad = 1;
    break;
  end

  col = 1; %Switch to the other column
 end %if col == 1

end %for i = 1:TotalRoll

%Now check to see if we would have overlapped any pallets or the aft of the
%acft.  If we did, then just return Xcurrent

if NumLoadedLeft > 0 && infeasLoad == 0
 if LoadedLeft(NumLoadedLeft,2) <= LeftAftFS
  Xcurrent (AcftIndex,1,1:NumLoadedLeft,1:4) = LoadedLeft(1:NumLoadedLeft,1:4);
 end
end
if NumLoadedRt > 0 && infeasLoad == 0
 if  LoadedRt(NumLoadedRt,2) <= RtAftFS
  Xcurrent (AcftIndex,2,1:NumLoadedRt,1:4) = LoadedRt(1:NumLoadedRt,1:4);
 end
end
```

```
  %Slide the CB to get things into CB limits if possible
  [Xcurrent,InCB] = SlideCBCenter (0,Xcurrent,AcftIndex,...
    AvailAcft,Cargo,CBLookup,NumLeftRoll,NumLeftPallet,LPallets,...
    NumRtRoll,NumRtPallet,RPallets,PalletTable);

  %Otherwise, we just return the original Xcurrent



end %CenterItems ~= [inf inf inf inf inf inf] && ...
```

Appendix D: Solution Representation

After completing the search, MPALPTS exports the cargo, available aircraft and current solution to a specific Excel File. This file has Visual Basic code which takes this data and builds a visual representation of the solution for each aircraft. The aircraft representation includes the exact location of each item within the cargo compartment as well as its width and weight. It also illustrates whether or not an item was centerline loaded. Grey lines in the cargo compartment display where any ramps meet the cargo floor. The tables below the figure display the acceptable and actual CB and ACL statistics as well as each item's axle weights and axle locations. This allows the user visually see how MPALPTS loaded the aircraft and to easily verify the load's feasibility. The visual representation depicted on the next page is the first C-5 from the M75 mixed test case.

C-5 9007



Axle Weights						
Item	1	2	3	4	5	6
29	488	5488	0	0	0	0
8	2845	2757	0	0	0	0
2	382	383	0	0	0	0
25	588	3758	3758	0	0	0
18	5488	5588	8888	8888	0	0
48	488	5588	0	0	0	0
22	2778	5592	0	0	0	0
5	736	7484	7368	0	0	0
32	0	0	0	0	0	0

TotalWt	151695
Max ACL	150000
CG	1332.53
CG Min	1288
CG Max	1398
Target CG	1332

Axle Locations						
Item	1	2	3	4	5	6
29	541	685	0	0	0	0
8	1822	1152	0	0	0	0
2	1252	1328	0	0	0	0
25	1334	1582	1558	0	0	0
18	742	758	846	854	0	0
48	1822	1884	0	0	0	0
22	1157	1329	0	0	0	0
5	1456	1515	1685	0	0	0
32	0	0	0	0	0	0

Appendix E: Specific Results

The following tables illustrate the specific results for each of the original six test cases. They show the number of cargo items placed on each aircraft and the aircraft's percentage of ACL and space used. It is interesting to note that scenarios with only C-17 aircraft tended to be limited by space while the C-5 and C-5/C-17 mixed problems tended to be limited by ACL. From the results, it appears AALPS does a very good job utilizing an aircraft's available space; however, MPALPTS dominates in finding solutions which are more limited by weight.

Table 14. P75 Mixed Results

AALPS				MPALPTS			
Acft	**Items**	**%ACL**	**%Space**	**Acft**	**Items**	**%ACL**	**%Space**
C-5	18	100	56	C-5	29	99.1	73.5
C-17	15	100	82	C-17	17	100.0	90.6
C-5	36	88	100	C-5	29	92.2	73.5
C-17	6	6	34	--	--	--	--
Totals/Avgs	**75**	**73.5**	**68**	**Totals/Avgs**	**75**	**97.1**	**79.2**

Table 15. P75 C-5 Results

AALPS				MPALPTS			
Acft	**Items**	**%ACL**	**%Space**	**Acft**	**Items**	**%ACL**	**%Space**
C-5	18	100	51	C-5	27	98.7	68.4
C-5	28	100	80	C-17	30	98.9	76.0
C-5	29	51.1	83	C-5	18	53.8	45.6
Totals/Avgs	**75**	**83.7**	**71.3**	**Totals/Avgs**	**75**	**83.7**	**63.4**

Table 16. P75 C-17 Results

AALPS				MPALPTS			
Acft	**Items**	**%ACL**	**%Space**	**Acft**	**Items**	**%ACL**	**%Space**
C-17	10	100	50	C-5	17	100.0	90.6
C-17	12	100	59	C-5	16	100.0	85.3
C-17	16	100	79	C-5	17	98.5	90.6
C-17	18	80	90	C-5	18	98.9	96.0
C-17	18	39	90	C-5	7	21.5	37.3
C-17	1	1	5	--	--	--	--
Totals/Avgs	**75**	**70.0**	**62.2**	**Totals/Avgs**	**75**	**83.8**	**80.0**

Table 17. P200 Mixed Results

AALPS				MPALPTS			
Acft	**Items**	**%ACL**	**%Space**	**Acft**	**Items**	**%ACL**	**%Space**
C-5	16	100	50	C-5	26	95.6	65.9
C-17	10	100	54	C-17	16	100.0	85.3
C-5	18	100	56	C-5	25	96.4	63.4
C-17	12	100	65	C-17	18	99.6	96.0
C-5	22	100	69	C-5	29	98.0	73.5
C-17	16	100	88	C-17	18	98.1	96.0
C-5	34	100	100	C-5	35	99.3	88.7
C-17	18	67	100	C-17	18	100.0	96.0
C-5	36	48	100	C-5	15	49.3	38.0
C-17	18	15	100	--	--	--	--
Totals/Avgs	**200**	**83.0**	**78.2**	**Totals/Avgs**	**200**	**92.9**	**78.1**

Table 18. P200 C-5 Results

AALPS				MPALPTS			
Acft	**Items**	**%ACL**	**%Space**	**Acft**	**Items**	**%ACL**	**%Space**
C-5	16	100	45	C-5	24	96.9	60.8
C-5	17	100	48	C-5	25	94.4	63.4
C-5	19	100	54	C-5	27	100.0	68.4
C-5	21	100	59	C-5	30	99.8	76.0
C-5	27	100	77	C-5	32	99.3	81.1
C-5	36	100	100	C-5	35	100.0	88.7
C-5	36	59	100	C-5	27	86.9	68.4
C-5	28	19	80	--	--	--	--
Totals/Avgs	**200**	**84.8**	**70.4**	**Totals/Avgs**	**200**	**96.8**	**72.4**

Table 19. P200 C-17 Results

AALPS				MPALPTS			
Acft	**Items**	**%ACL**	**%Space**	**Acft**	**Items**	**%ACL**	**%Space**
C-17	10	100	50	C-17	17	98.9	90.6
C-17	10	100	50	C-17	17	97.4	90.6
C-17	10	100	49	C-17	17	100.0	90.6
C-17	11	100	54	C-17	17	98.5	90.6
C-17	12	100	59	C-17	18	95.6	96.0
C-17	12	100	59	C-17	17	98.5	90.6
C-17	14	100	69	C-17	18	99.6	96.0
C-17	16	100	79	C-17	18	99.6	96.0
C-17	18	96	90	C-17	17	95.6	90.6
C-17	18	83	90	C-17	18	100.0	96.0
C-17	18	63	90	C-17	17	93.7	90.6
C-17	18	45	90	C-17	9	51.5	48.0
C-17	18	31	90	--	--	--	--
C-17	15	11	75	--	--	--	--
Totals/Avgs	**200**	**88.7**	**72.9**	**Totals/Avgs**	**200**	**98.2**	**88.8**

Table 20. R75 Mixed Results

AALPS				MPALPTS			
Acft	Items	%ACL	%Space	Acft	Items	%ACL	%Space
C-5	6	100	49	C-5	5	89.6	85.8
C-17	4	100	50	C-17	4	99.4	94.0
C-5	4	91	32	C-5	6	89.3	80.7
C-17	3	100	46	C-17	3	89.2	90.7
C-5	6	88	57	C-5	11	95.4	84.6
C-17	4	99	49	C-17	6	90.4	90.3
C-5	8	61	46	C-5	3	97.5	95.9
C-17	4	99	51	C-17	10	99.5	89.1
C-5	11	69	63	C-5	7	68.8	85.5
C-17	3	53	44	C-17	13	51.2	87.4
C-5	10	53	71	C-5	7	67.6	90.4
C-17	5	36	55	--	--	--	--
C-5	7	15	34	--	--	--	--
Totals/Avgs	**75**	**74.2**	**49.8**	**Totals/Avgs**	**75**	**85.3**	**88.6**

Table 21. R75 C-5 Results

AALPS				MPALPTS			
Acft	**Items**	**%ACL**	**%Space**	**Acft**	**Items**	**%ACL**	**%Space**
C-5	4	100	42	C-5	5	95.9	87.4
C-5	4	94	32	C-5	11	95.3	84.0
C-5	5	100	40	C-5	6	98.2	98.7
C-5	7	87	57	C-5	9	53.8	85.2
C-5	7	100	47	C-5	7	96.2	93.3
C-5	4	38	20	C-5	6	74.4	80.7
C-5	11	83	63	C-5	9	97.0	91.5
C-5	9	61	51	C-5	13	79.6	92.6
C-5	11	60	69	C-5	9	78.4	85.7
C-5	12	44	66	--	--	--	--
C-5	1	2	6	--	--	--	--
Totals/Avgs	75	69.9	44.8	Totals/Avgs	75	85.4	88.8

Table 22. R75 C-17 Results

AALPS				MPALPTS			
Acft	**Items**	**%ACL**	**%Space**	**Acft**	**Items**	**%ACL**	**%Space**
C-17	3	91.2	80.7	C-17	3	91.2	80.7
C-17	4	97.0	91.2	C-17	4	97.0	91.2
C-17	4	93.4	52.4	C-17	4	93.4	52.4
C-17	3	67.9	91.3	C-17	3	67.9	91.3
C-17	3	63.6	79.3	C-17	3	63.6	79.3
C-17	3	72.1	91.7	C-17	3	72.1	91.7
C-17	3	72.1	91.7	C-17	3	72.1	91.7
C-17	3	63.4	95.7	C-17	3	63.4	95.7
C-17	3	78.5	96.1	C-17	3	78.5	96.1
C-17	4	98.0	88.5	C-17	4	98.0	88.5
C-17	5	67.3	95.7	C-17	5	67.3	95.7
C-17	4	85.0	89.8	C-17	4	85.0	89.8
C-17	7	93.0	82.3	C-17	7	93.0	82.3
C-17	7	63.8	92.3	C-17	7	63.8	92.3
C-17	5	78.1	83.3	C-17	5	78.1	83.3
C-17	6	68.4	92.1	C-17	6	68.4	92.1
C-17	8	28.6	90.9	C-17	8	28.6	90.9
Totals/Avgs	**75**	**75.4**	**87.3**	**Totals/Avgs**	**75**	**75.4**	**87.3**

Table 23. R200 Mixed Results

AALPS				MPALPTS			
Acft	**Items**	**%ACL**	**%Space**	**Acft**	**Items**	**%ACL**	**%Space**
C-5	7	100	47	C-5	7	94.5	98.9
C-17	4	100	52	C-17	3	93.8	91.7
C-5	7	100	47	C-5	6	98.2	81.4
C-17	4	99	52	C-17	3	93.8	91.7
C-5	6	100	40	C-5	7	91.3	86.4
C-17	4	100	45	C-17	3	99.8	46.9
C-5	6	100	40	C-5	10	91.1	81.4
C-17	4	100	45	C-17	3	93.8	91.7
C-5	3	90	30	C-5	8	88.7	94.1
C-17	2	96	35	C-17	6	99.6	95.6
C-5	4	91	32	C-5	7	99.5	90.5
C-17	3	100	46	C-17	4	98.5	92.5
C-5	5	100	42	C-5	7	95.3	82.1
C-17	3	100	46	C-17	4	99.6	95.1
C-5	5	100	46	C-5	8	99.3	97.4
C-17	4	100	61	C-17	4	99.7	91.3
C-5	6	85	60	C-17	6	99.6	93.9
C-17	4	99	61	C-5	10	80.3	96.1
C-5	5	89	40	C-17	4	97.3	92.4
C-17	4	99	51	C-5	8	92.1	96.9
C-5	4	38	21	C-17	3	97.5	47.9
C-17	4	99	51	C-5	13	79.9	93.5
C-5	7	68	54	C-5	12	54.5	82.9
C-17	4	99	51	C-17	6	99.6	94.6
C-5	11	82	68	C-5	11	94.7	79.3
C-17	3	53	44	C-17	7	99.9	88.9
C-5	13	72	73	C-5	16	92.5	86.0
C-17	3	53	44	C-17	6	98.0	96.3
C-5	12	57	70	C-5	8	92.5	95.9
C-17	5	53	72	C-17	5	99.8	85.9
C-5	15	42	66	C-5	8	91.6	79.1
C-17	8	25	59	--	--	--	--
C-5	15	23	64	--	--	--	--
C-17	6	9	36	--	--	--	--
Totals/Avgs	**200**	**80.0**	**49.7**	**Totals/Avgs**	**200**	**93.6**	**88.1**

Table 24. R200 C-5 Results

AALPS				MPALPTS			
Acft	**Items**	**%ACL**	**%Space**	**Acft**	**Items**	**%ACL**	**%Space**
C-5	4	100	42	C-5	6	95.9	95.6
C-5	4	100	43	C-5	7	98.4	88.0
C-5	4	100	43	C-5	8	99.7	94.3
C-5	6	100	39	C-5	8	98.4	67.7
C-5	6	100	39	C-5	8	94.7	69.0
C-5	6	100	39	C-5	9	79.2	73.2
C-5	3	90	28	C-5	6	90.7	85.5
C-5	3	90	28	C-5	7	99.1	86.4
C-5	6	100	42	C-5	7	94.4	80.4
C-5	5	100	40	C-5	7	86.2	83.4
C-5	5	100	40	C-5	8	93.6	87.5
C-5	5	100	51	C-5	8	91.7	85.1
C-5	7	87	61	C-5	11	76.1	96.6
C-5	11	89	64	C-5	10	95.4	84.1
C-5	6	100	44	C-5	6	95.1	86.6
C-5	6	100	41	C-5	5	98.0	86.5
C-5	7	100	39	C-5	6	88.6	83.2
C-5	4		20	C-5	7	85.6	85.5
C-5	10	77	59	C-5	5	76.2	84.0
C-5	13	97	72	C-5	11	68.0	89.3
C-5	10	55	51	C-5	11	85.8	91.7
C-5	12	69	63	C-5	12	84.6	82.3
C-5	12	65	67	C-5	12	99.1	93.0
C-5	13	51	74	C-5	15	92.4	88.5
C-5	16	40	63	--	--	--	--
C-5	14	20	62	--	--	--	--
C-5	2	2	6	--	--	--	--
Totals/Avgs	**200**	**80.4**	**46.7**	**Totals/Avgs**	**200**	**90.3**	**85.3**

Table 25. R200 C-17 Results

AALPS				MPALPTS			
Acft	Items	%ACL	%Space	Acft	Items	%ACL	%Space
C-17	4	100	52	C-17	3	100.0	92.0
C-17	4	100	52	C-17	3	95.3	89.3
C-17	4	100	52	C-17	3	99.8	79.4
C-17	4	100	52	C-17	3	97.6	94.0
C-17	4	100	52	C-17	3	91.2	80.7
C-17	4	99	52	C-17	3	85.1	81.0
C-17	4	100	45	C-17	3	91.2	96.3
C-17	5	100	50	C-17	3	91.2	96.3
C-17	5	100	50	C-17	3	91.2	96.3
C-17	5	100	50	C-17	3	91.0	80.3
C-17	5	100	50	C-17	4	96.1	89.8
C-17	5	100	50	C-17	4	95.5	92.2
C-17	2	96	35	C-17	3	66.0	73.8
C-17	2	96	35	C-17	3	73.5	93.5
C-17	2	96	35	C-17	4	95.9	92.7
C-17	2	96	35	C-17	3	83.9	90.2
C-17	2	96	35	C-17	6	95.1	95.4
C-17	2	96	35	C-17	5	88.7	86.8
C-17	3	100	46	C-17	4	99.5	92.7
C-17	3	100	46	C-17	4	66.9	49.6
C-17	4	98	54	C-17	4	77.1	88.2
C-17	4	97	54	C-17	5	78.7	88.8
C-17	4	97	54	C-17	4	88.8	82.3
C-17	4	97	54	C-17	4	76.4	84.8
C-17	4	99	61	C-17	5	65.4	86.7
C-17	4	99	61	C-17	5	79.0	91.6
C-17	4	100	62	C-17	3	78.5	96.1
C-17	3	92	37	C-17	4	78.0	58.9
C-17	2	52	21	C-17	4	78.8	59.8
C-17	3	74	51	C-17	4	70.3	56.6
C-17	3	74	51	C-17	4	72.9	60.9
C-17	4	85	59	C-17	4	69.1	88.9
C-17	8	100	82	C-17	6	80.7	94.4
C-17	6	97	77	C-17	6	71.4	87.7
C-17	6	89	74	C-17	6	74.8	91.7
C-17	8	67	75	C-17	7	79.0	79.8
C-17	8	56	76	C-17	6	68.8	85.3
C-17	8	53	88	C-17	7	88.4	76.7

Table 25 (Continued). R200 C-17 Results

AALPS				MPALPTS			
Acft	**Items**	**%ACL**	**%Space**	**Acft**	**Items**	**%ACL**	**%Space**
C-17	8	38	70	C-17	3	88.5	77.2
C-17	8	25	59	C-17	8	77.0	92.3
C-17	8	22	61	C-17	6	98.4	93.9
C-17	8	20	58	C-17	6	91.3	93.9
C-17	9	14	53	C-17	12	27.5	95.4
C-17	1	1	6	C-17	7	58.2	92.9
Totals/Avgs	**200**	**82.3**	**52.4**	**Totals/Avgs**	**200.0**	**82.1**	**85.2**

Table 26. M75 Mixed Results

AALPS				MPALPTS			
Acft	**Items**	**%ACL**	**%Space**	**Acft**	**Items**	**%ACL**	**%Space**
C-5	9	100	46	C-5	9	87.8	78.7
C-17	2	96	35	C-17	6	99.2	96.0
C-5	7	80	60	C-5	10	100.0	76.1
C-17	5	96	62	C-17	4	84.3	92.9
C-5	11	90	67	C-5	9	77.5	62.0
C-17	6	75	61	C-17	7	75.1	83.9
C-5	10	27	38	C-5	30	97.6	76.0
C-17	18	100	100	--	--	--	--
C-17	7	7	22	--	--	--	--
Totals/Avgs	**75**	**74.6**	**54.6**	**Totals/Avgs**	**75**	**88.8**	**80.8**

Table 27. M75 C-5 Results

AALPS				MPALPTS			
Acft	**Items**	**%ACL**	**%Space**	**Acft**	**Items**	**%ACL**	**%Space**
C-5	9	100	46	C-5	10	81.6	94.2
C-5	9	99	52	C-5	6	87.1	70.9
C-5	7	78	61	C-5	8	95.6	88.9
C-5	9	79	58	C-5	7	82.8	91.4
C-5	9	61	51	C-5	20	96.6	76.6
C-5	4	20	21	C-5	24	74.3	85.6
C-5	28	80	89	--	--	--	--
Totals/Avgs	**75**	**73.9**	**54.0**	**Totals/Avgs**	**75**	**86.3**	**84.6**

Table 28. M75 C-17 Results

AALPS				MPALPTS			
Acft	**Items**	**%ACL**	**%Space**	**Acft**	**Items**	**%ACL**	**%Space**
C-17	6	99.4	50.0	C-17	4	99.8	91.5
C-17	3	99.5	41.0	C-17	5	85.6	93.5
C-17	5	99.9	51.0	C-17	6	96.1	93.4
C-17	6	99.8	65.0	C-17	3	72.3	89.8
C-17	4	86.4	62.0	C-17	4	76.8	95.1
C-17	5	95.0	64.0	C-17	5	71.8	94.0
C-17	8	71.9	78.0	C-17	7	64.4	98.2
C-17	8	87.1	81.0	C-17	5	66.0	87.6
C-17	10	55.1	57.0	C-17	6	86.1	90.1
C-17	18	68.1	100.0	C-17	17	99.3	90.6
C-17	2	1.1	11.0	C-17	13	45.2	69.3
Totals/Avgs	**75**	**78.5**	**60.0**	**Totals/Avgs**	**75**	**78.5**	**90.3**

Table 29. M200 Mixed Results

AALPS				MPALPTS			
Acft	**Items**	**%ACL**	**%Space**	**Acft**	**Items**	**%ACL**	**%Space**
C-5	7	100	39	C-5	12	97.8	89.6
C-17	3	100	41	C-17	6	96.5	95.3
C-5	5	100	36	C-5	6	93.3	92.3
C-17	3	100	41	C-17	3	92.3	99.2
C-5	5	100	42	C-5	8	92.8	84.7
C-17	4	100	61	C-17	6	99.6	91.9
C-5	9	95	62	C-5	9	99.2	67.1
C-17	6	100	56	C-17	6	97.6	67.3
C-5	11	100	55	C-5	8	96.9	84.4
C-17	5	100	60	C-17	6	80.8	94.0
C-5	13	83	73	C-5	11	70.9	98.0
C-17	6	56	58	C-17	7	98.0	94.0
C-5	14	49	72	C-5	15	85.9	83.5
C-17	8	31	53	C-17	17	99.6	90.6
C-5	8	33	31	C-5	29	96.0	73.5
C-17	11	100	61	C-17	17	99.6	90.6
C-5	23	100	72	C-5	34	90.0	86.2
C-17	18	97	100	--	--	--	--
C-5	36	52	100	--	--	--	--
C-17	5	3	28	--	--	--	--
Totals/Avgs	**200**	**80.0**	**57.1**	**Totals/Avgs**	**200**	**93.3**	**87.2**

Table 30. M200 C-5 Results

AALPS				MPALPTS			
Acft	**Items**	**%ACL**	**%Space**	**Acft**	**Items**	**%ACL**	**%Space**
C-5	7	100	37	C-5	7	97.1	60.6
C-5	5	100	34	C-5	6	99.7	80.3
C-5	6	100	36	C-5	9	88.6	88.8
C-5	5	100	40	C-5	9	96.2	85.1
C-5	5	100	44	C-5	9	96.1	88.8
C-5	10	88	62	C-5	6	89.2	85.6
C-5	6	99	46	C-5	9	92.8	84.6
C-5	12	100	66	C-5	9	93.6	86.9
C-5	13	75	66	C-5	13	78.1	73.2
C-5	12	56	67	C-5	12	87.5	85.0
C-5	14	29	57	C-5	22	95.1	71.1
C-5	17	100	48	C-5	31	100.0	78.6
C-5	23	100	65	C-5	35	99.1	88.7
C-5	36	100	100	C-5	23	68.0	58.3
C-5	29	33	83	--	--	--	--
Totals/Avgs	**200**	**85.3**	**56.7**	**Totals/Avgs**	**200**	**91.5**	**79.7**

Table 31. M200 C-17 Results

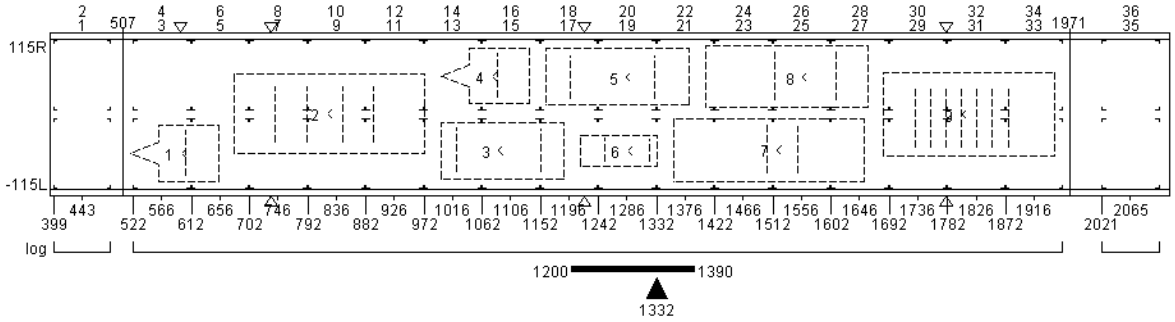
AALPS				MPALPTS			
Acft	**Items**	**%ACL**	**%Space**	**Acft**	**Items**	**%ACL**	**%Space**
C-17	7	99.9	49.0	C-17	5	97.3	97.4
C-17	3	99.9	39.0	C-17	5	87.2	92.0
C-17	3	99.9	39.0	C-17	6	98.8	89.0
C-17	3	99.9	39.0	C-17	5	95.7	80.4
C-17	3	99.5	39.0	C-17	3	60.3	43.6
C-17	3	99.5	39.0	C-17	6	94.8	90.3
C-17	3	99.5	36.0	C-17	6	88.7	86.5
C-17	5	94.3	52.0	C-17	7	89.9	91.1
C-17	4	98.6	59.0	C-17	6	92.8	94.5
C-17	4	98.6	59.0	C-17	6	74.6	93.2
C-17	5	99.8	50.0	C-17	6	79.1	95.5
C-17	5	99.8	40.0	C-17	4	80.0	85.6
C-17	6	89.3	65.0	C-17	4	68.5	92.7
C-17	8	95.0	80.0	C-17	4	87.1	94.4
C-17	8	87.7	77.0	C-17	4	89.6	90.8
C-17	8	58.3	80.0	C-17	6	79.5	92.8
C-17	11	51.2	73.0	C-17	7	67.4	89.1
C-17	7	30.8	47.0	C-17	6	78.8	92.5
C-17	13	99.9	77.0	C-17	7	83.2	74.9
C-17	12	100.0	60.0	C-17	8	100.0	86.5
C-17	14	100.0	69.0	C-17	18	99.3	96.0
C-17	17	100.0	85.0	C-17	17	98.5	90.6
C-17	18	77.8	90.0	C-17	18	99.6	96.0
C-17	18	42.2	90.0	C-17	18	96.3	96.0
C-17	12	11.9	60.0	C-17	18	48.1	96.0
Totals/Avgs	**200**	**85.4**	**59.7**	**Totals/Avgs**	**200**	**85.4**	**89.1**

Appendix F: MPALPTS Load Plans

The first load plan presented represents the solution MPALPTS generated for the first C-5 aircraft loaded from the M75 mixed test set. The exact solution representation was manually loaded in AALPS. MPALPTS computed the CB at FS1332.53, and AALPS computed it at FS1332. The second load plan represents the third C-17 MPALPTS loaded from the M75 mixed test set and includes both rolling stock and pallets. MPALPTS computed this load's CB at FS857.23, and AALPS computed it at FS857. If any load violations were present, they would have been identified in the Flags/Warnings section of the load plan.

Aircraft type/Config	**: C-5/STD-AL**	**Mission Type**	**: Channel**
Delivery method	**: AL**	**Mission #**	**:**
Unit Being Airlifted	**:**	**Aircraft tail #**	**:**
Type movement plan	**:**	**System chalk #**	**:**
Departure date & time	**:**		
Departure airfield	**:**		
Destination airfield	**:**		
Load Description	**:**		

MAIN DECK



SQ/D	Model/Nomenclature	LEN	WDT	HT	WT	FSN	TSN	CB	HZ	FL	V	D	SH	CCC
1/M	M1101/CHASSIS TRAILER	137	87	73	3500	517	654	596	n	E	N	P		R2D
2/M	M1134/ANTI-TANK VEH/ST	294	125	142	41160	679	973	822	n	E	N	P		R0D
3/M	M1097/TRK, UTIL, HVY,	191	86	72	5600	998	1189	1086	n	E	N	P		R2D
4/M	M1101/CHASSIS TRAILER	137	87	73	3500	998	1135	1077	n	E	N	P		R2D
5/M	M1008A1/TRUCK CARGO TA	223	87	96	8170	1160	1383	1284	n	E	N	P		R2D
6/M	6000 LB/TRAILER PLATFO	119	48	27	765	1214	1333	1286	n	E	N	P		A2B
7/M	HP-15T/TRAILER FLATBED	294	96	67	8000	1358	1652	1518	n	E	N	P	P	R2D
8/M	FLU419/TRACTOR, ALL-WH	250	96	102	15920	1408	1658	1557	y	E	N	P		R2D
9/M	M992/CARRIER AMMO TRKD	265	130	115	45080	1683	1948	1803	y	E	N	P	PR	A1D

Total # of Pax: 0 Weight/Pax: 210 Total Pax Weight: 0
Total # of Subfloors: 0 Weight/Subfloor: 0 Total Subfloor Weight: 0
Total Cargo Wt: 131695 Total Load Wt: 131695 ACL: 150000
%ACL: 88 %ZF: 0 Load CB: 1332

SQ/D Flags/Warnings

SQ/D	Class/Zone
3/M	9
5/M	9
8/M	9
9/M	9

ALL HAZARDOUS MATERIALS COVERED BY THIS MANIFEST HAVE BEEN INSPECTED AND FOUND TO BE PACKAGED IN THE PROPER OUTSIDE CONTAINER, FREE OF VISIBLE DAMAGE AND LEAKS AND IS PROPERLY CERTIFIED

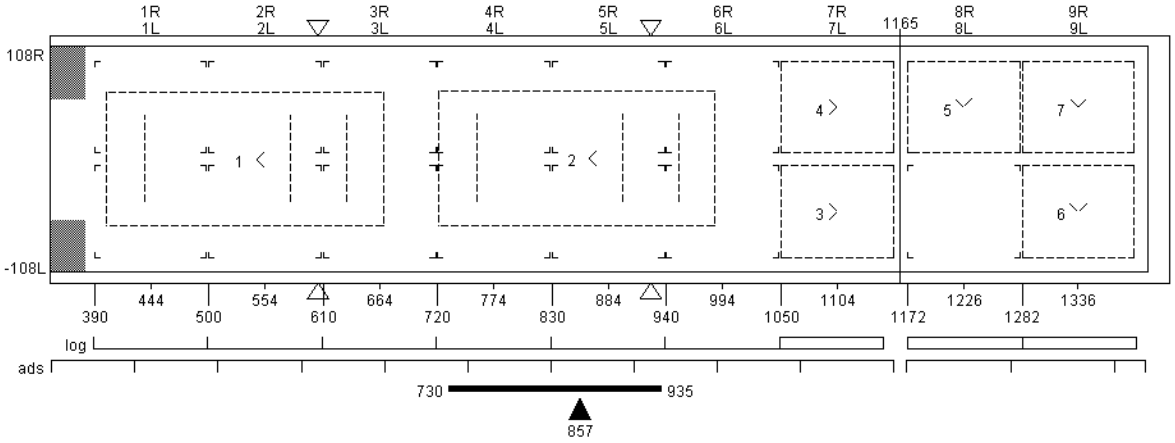
_____
Air Terminal Representative Signature

I HAVE BEEN BRIEFED ACCORDING TO AFMAN 24-204(I), PARAGRAPH 1.2.9, ON HAZARDOUS CARGO COVERED BY THIS MANIFEST

_____
Aircraft Crewmember Signature

Aircraft type/Config	**: C-17/STD-AL**	**Mission Type**	**:**
Delivery method	**: AL**	**Mission #**	**:**
Unit Being Airlifted	**:**	**Aircraft tail #**	**:**
Type movement plan	**:**	**System chalk #**	**:**
Departure date & time	**:**		
Departure airfield	**:**		
Destination airfield	**:**		
Load Description	**:**		

MAIN DECK



SQ/D	Model/Nomenclature	LEN	WDT	HT	WT	FSN	TSN	CB	HZ	FL	V	D	SH	CCC
1/M	00-M51A1/TRUCK DUMP 5-	266	128	112	22146	402	668	546	n	E	N	P		R1D
2/M	00-M51A1/TRUCK DUMP 5-	266	128	112	22146	721	987	865	n	E	N	P		R1D
3/M	7333 x 83/PALLET, 463L	108	88	83	7333	1050	1158	1104	n	E	N	P		J3B
4/M	9667 x 64/PALLET, 463L	108	88	64	9667	1050	1158	1104	n	E	N	P		J3B
5/M	4667 x 37/PALLET, 463L	88	108	37	4667	1172	1280	1226	n	E	N	P		J3B
6/M	1000 x 38/PALLET, 463L	88	108	38	1000	1282	1390	1336	n	E	N	P		J3B
7/M	667 x 30/PALLET, 463L	88	108	30	667	1282	1390	1336	n	E	N	P		J3B

Total # of Pax: 0 Weight/Pax: 210 Total Pax Weight: 0
Total # of Subfloors: 0 Weight/Subfloor: 0 Total Subfloor Weight: 0
Total Cargo Wt: 67626 Total Load Wt: 67626 ACL: 90000
%ACL: 75 %ZF: 0 Load CB: 857

SQ/D Flags/Warnings

SQ/D Class/Zone

ALL HAZARDOUS MATERIALS COVERED BY THIS MANIFEST HAVE BEEN INSPECTED AND FOUND TO BE PACKAGED IN THE PROPER OUTSIDE CONTAINER, FREE OF VISIBLE DAMAGE AND LEAKS AND IS PROPERLY CERTIFIED

_____
Air Terminal Representative Signature

I HAVE BEEN BRIEFED ACCORDING TO AFMAN 24-204(I), PARAGRAPH 1.2.9, ON HAZARDOUS CARGO COVERED BY THIS MANIFEST

_____
Aircraft Crewmember Signature

Appendix G: Blue Dart

Blue Dart Submission Form

First Name: Robert Last Name: Nance

Rank (Military, AD, etc.): Major Designator # AFIT/BD/ENS/09-11

Student's Involved in Research for Blue Dart: Maj Robert Nance

Position/Title: C-5 Evaluator Pilot / AFIT Master's Student

Phone Number: DSN 225-3636 E-mail: robert.nance@af.mil

School/Organization: AFIT/ENS

Status: [X] Student [ ] Faculty [ ] Staff [ ] Other

Optimal Media Outlet (optional): _____

Optimal Time of Publication (optional): _____

General Category / Classification:

[ ] core values	[ ] command	[ ] strategy
[ ] war on terror	[ ] culture & language	[ ] leadership & ethics
[ ] warfighting	[ ] international security	[ ] doctrine

[X] other (specify): Military Airlift

Suggested Headline: Improving the efficiency of USAF airlift

Keywords: airlift, efficiency, logistics, Air Mobility Command

Blue Dart

As mentioned by the JCS Director of Logistics, Lt Gen Gainey, during her recent visit to AFIT, there are two general measures of a logistics system: effectiveness and efficiency.

To be effective, the system must deliver the required goods to the proper recipients on time. Efficient systems deliver goods in the most economical manner. Lt Gen Gainey noted that effectiveness is more important than efficiency in a military supply system; however, in today's budget constrained environment, efficiency is clearly an important goal particularly when it comes to the most expensive mode of transportation: airlift. According to AMC/A9 data, the US Air Force paid $22,998 and $12,911 per flight hour in FY08 to utilize C-5 and C-17 aircraft, respectively. If we assume approximately 30 hours of flight time are required to fly CONUS to the Middle East and back, each C-5 mission costs roughly $689,940 and each C-17 mission costs $387,330. Furthermore, AMC/A9 data indicates that the 12,760 C-5 and C-17 operational mission *sorties* from 1 Jan 08 to 30 Sep 08 only used, on average, approximately one-third of their available cargo capacity. (This data measures the actual versus planning cargo weights for the C-5 and C-17 for each sortie of a mission and includes empty pre- and de-positioning mission sorties). This clearly indicates the USAF operates an *inefficient* airlift system.

Why is USAF airlift inefficient? The answer is simple: it is very difficult to optimally load aircraft! Given a ramp full of pallets and equipment, determining which items to place on which aircraft and where exactly to place them has billions of possible combinations. This is one of the factors which drove the DoD to invest in the Automated Airlift Load Planning Software (AALPS). Given cargo and available aircraft, AALPS automatically generates load plans and helps load planners solve this difficult problem.

While AALPS is a very powerful tool, it has been found to produce inefficient loadings. My thesis research is focused on creating a better loading algorithm which uses fewer aircraft than AALPS. This algorithm, called the Mixed Payload Airlift Load Planning Tabu Search (MPALPTS), loads a given set of cargo (pallets and rolling stock) into C-5 and/or C-17 aircraft and makes a limited number of assumptions. Therefore, it is operationally useful. In the 20 test scenarios (which included pallet only, rolling stock only and a mix of pallet and rolling stock loads on C-5 and/or C-17 aircraft), my algorithm achieved an average aircraft reduction of 11% when compared to AALPS solutions on the same cargo. A subset of MPALPTS solutions was manually loaded into AALPS to verify feasibility in the loadings. I found nearly all of MPALPTS solutions were feasible, and the few that were not required only very small adjustments (to clear the C-5's troop compartment ladder, for example). However, MPALPTS did take much longer than AALPS to find these improved solutions.

From Feb 07 to Jan 08, AMC reported there were 686 C-5 and 1551 C-17 multi-leg operational missions. Assuming all of the missions were originally planned with AALPS and were reloaded with MPALPTS which increased efficiency by 11%, AMC would have flown 75 fewer C-5 missions and 171 fewer C-17 missions. If each mission averaged 30 flight hours, this equates to a savings of $117,978,930 in a 12 month period with zero impact on effectiveness. The DoD should incorporate MPALPTS into the current version of AALPS to increase USAF airlift efficiency.

The views expressed in this article are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the US Government.

Feb 09

Bibliography

Aarts, Emile, and Jan Karel Lenstra. *Local Search in Combinatorial Optimization.* West Sussex, England: John Wiley & Sons, 1997.

AFDD 1. *Air Force Basic Doctrine, Complement of Joint Publication 1: Joint Warfare of the Armed Forces of the United States.* 2003.

AFDD 2-6. *Air Mobility Operations.* 2006.

Air Mobility Command. "AMC Affiliation Workbook 36-101 Vol 2 Airlift Planners Course." October 1, 2004.

"T.O. 1-C-17A-9 Loading Instructions." November 15, 2007, Secretary of the Air Force, Washington DC.

"T.O. 1-C-5A-9 Loading Instructions Manual." Feburary 1, 2003, Secretary of the Air Force, Washington DC.

Anderson, Don, LtCol (Ret.), interview by R. Larry Nance. *AMC Airlift Data from GATES* (September 15, 2008).

Brenneman, William A., and William R. Myers. "Robust Parameter Design with Categorical Noise Variables." *Journal of Quality Design*, 2003: 335-341.

Chocolaad, Christopher A. "Solving Geometric Knapsack PRoblems using Tabu Search Heuristics." Master's Thesis, Air Force Institute of Technology, 1998.

Garey, Michael R., and David S. Johnson. *Computers and Intractability.* New York: W.H. Freeman and Company, 1979.

Gehring, H. et al. "A Computer-Based Heuristic for Packing Pooled Shipment Containers." *European Journal of Operations Research*, 1990: 227-288.

Glover, Fred. "Tabu Search--Part I." *Operations Research Society of America* 1, no. 3 (1989): 190-207.

Goulimis, Constantine N. "ASP, The Art and Science of Practice: Appeal to NC-Completeness Considered Harmful: Does the Fact That a Problem Is NP-Complete Tell Us Anything?" *Interfaces* 37, no. 6 (November-December 2007): 584-586.

Harwig, J. M., J. W. Barnes, and J. T. Moore. "An Adaptinve Tabu Search Approach for 2-Dimensional Orthogonal Packing Problems." *Military Operations Research* 11, no. 2 (2006): 5-26.

Heidelberg, K.R. *A Bin Packing Algorithm for Cargo Conveyance Systems.* Master's Thesis, Virginia Commonwealth University, 1995.

Heidelberg, Kurt R., Gregory S. Parnell, and James E. Ames IV. "Automated Air Load Planning." *Naval Research Logistics* 45 (1998): 751-768.

Herbison, LtCol Dave, interview by Maj R. Larry Nance. *Air Mobility Command A9 Deptuty Director* (September 16, 2008).

Hiremath, Chaitr S., and Raymond R. Hill. "New Greedy Heuristic for the Multiple-Choice Multi-Dimensional Knapsack Problem." *International Journal of Operational Research* 2, no. 4 (2007): 495-511.

Huang, J.H. *Sun Tzu, The New Translation.* New York: William Morrow, 1993.

Lodi, A., S. Martello, and D. Vigo. "Recent advances on two-dimensional bin packing problems." *Discrete Applied Mathematics*, 2002: 379-396.

Misevicius, A. "Using Iterated Tabu Search for the Traveling Salesman Problem." *Informacines Technologijos IR Valdymas*, 2004: 29-40.

Ng, Kevin Y. K. "A Multicriteria Optimization Approach to Aircraft Loading." *Operations Research* 40, no. 6 (November-December 1992): 1200-1205.

Pearl, Judea. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Reading, Massachussets: Addison-Wesley Publishing Company, 1984.

Roesener, August G. "An Advanced Tabu Search Approach To The Airlift Loading Problem." PhD Dissertation, 2006.

Romaine, Johathan M. "Solving the multidimensional multiple knapsack problem with packing constraints using tabu search." Master's Thesis, Air Force Institute of Technology, 1999.

USAF. *Air Force Factsheets.* April 2008. http://www.af.mil/factsheets/factsheet.asp (accessed July 15, 2008).

USTRANSCOM. *www.ustranscom.mil.* August 2007. http://www.transcom.mil/j5/pt/dtrpart3/dtr_part_iii_app_i.pdf (accessed July 11, 2008).

Vazirani, Vijay V. *Approximation Algorithms.* Heidelberg, Germany: Springer, 2003.

Wakefield, Senior Master Sergeant Tim, interview by Maj R. Larry Nance. *Chief C-5 Loadmaster, Headquarters Air Mobility Command* (September 15, 2008).

Woosley, Laurence A. *Integer Programming.* New York: John Wiley and Sons, Inc., 1998.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From – To)*
03-26-2009	**Master's Thesis**	Aug 2007 – Mar 2009

4. TITLE AND SUBTITLE

An Advanced Tabu Search Approach to Solving the Mixed Payload Airlift Load Planning Problem

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Nance, Robert L., Maj, 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 Hobson Street, Building 642
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GOR/ENS/09-11

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
HQ AMC/A9
Attn: Mr. Dave Merrill
402 Scott Drive Unit 3M12
Scott AFB, IL 62225

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

This paper presents a new tabu search based two-dimensional bin packing algorithm which produces high quality solutions to the Mixed Payload Airlift Load Planning (MPALP) problem using C-5 and C-17 aircraft. This algorithm, called Mixed Payload Airlift Load Planning Tabu Search (MPALPTS), surpasses previous research conducted in this area because, in addition to pure pallet cargo loads, MPALPTS can accommodate rolling stock cargo (i.e. tanks, trucks, HMMMVs, etc.) while still maintaining aircraft feasibility with respect to aircraft center of balance, mandatory cargo separations, aircraft floor structural limitations, etc. Furthermore, while this research is currently restricted to C-5 and C-17 aircraft, MPALPTS is capable of modeling nearly any type of cargo aircraft and requires a limited number of assumptions thereby making it applicable to operational missions. To demonstrate its effectiveness, the load plans generated by MPALPTS are directly compared to those generated by the Automated Air Load Planning Software (AALPS) for a given cargo set; AALPS is the load planning software currently mandated for use in all Department of Defense load planning. While more time consuming than AALPS, MPALPTS required the same or fewer aircraft than AALPS in all test scenarios.

15. SUBJECT TERMS
Airlift, Bin Packing, Tabu Search, Air Mobility Command, Aircraft Loading Problem, AALPS

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			August G. Roesener, Maj, USAF (ENS)
U	**U**	**U**	**UU**	149	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-3636, ext 4314; e-mail: August.Roesener@afit.edu