AFIT/GOR/ENS/94M-05

A HEURISTIC APPROACH TO DETERMINING CARGO FLOW AND SCHEDULING FOR AIR MOBILITY COMMAND'S CHANNEL CARGO SYSTEM

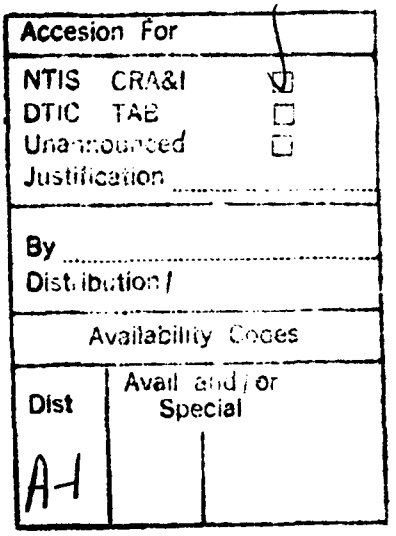
THESIS

John Fitzsimmons Jr., Captain, U.S. Air Force
John Walker, Captain, U.S. Air Force

AFIT/GOR/ENS/94M-05

Approved for public release; distribution unlimited

The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.



AFIT/GOR/ENS/94M-05

A HEURISTIC APPROACH TO DETERMINING CARGO FLOW AND SCHEDULING FOR AIR MOBILITY COMMAND'S CHANNEL CARGO SYSTEM

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Operations Research

John Fitzsimmons Jr., B.S.

Captain, U.S. Air Force

John Walker, B.A.

Captain, U.S. Air Force

March 1994

Thesis Approval

STUDENTS: Captain John Fitzsimmons Jr. CLASS: GOR-94M
Captain John Walker

THESIS TITLE: A Heuristic Approach to Determining Cargo Flow and Scheduling for Air Mobility Command's Channel Cargo System

DEFENSE DATE: 25 February 1994

COMMITTEE:	NAME/DEPARTMENT	SIGNATURE
Co-advisor	LTC James T. Moore/ENS	James T. Moore
Co-advisor	MAJ John J. Borsi/ENS	John J. Borsi

Acknowledgements

We are indebted to many people who provided their assistance in this research. The process could not have started, progressed, and ended without the guidance and patience of our advisors, LTC James T. Moore and MAJ John J. Borsi. They made it blindingly clear that the thesis process can be completed with persistence and a bit of perspiration. We are also grateful to several folks at the AMC Force Structure Analysis Office: 1LT Jonathan Robinson and Mr. Alan Whisman. They answered our endless questions and data requests punctually and professionally. Additionally, thanks goes to Captain Jean Steppe, who took time out of her already hectic schedule to provide much-needed feedback. We would also like to express our appreciation to our families, whose constant support helped us maintain a positive outlook.

John Fitzsimmons Jr.
John Walker

Table of Contents

List of Figures

List of Tables

Abstract

This research investigated a heuristic approach to schedule aircraft for the channel cargo system of the United States Air Force's Air Mobility Command (AMC). Given cargo/frequency of visit requirements, a fleet of aircraft, and possible routes, the objective of this research was to develop, implement, and test an iterative procedure to efficiently schedule and load aircraft in order to maximize the flow of cargo through the channel cargo system. Once a level of flow was established, attempts were made to minimize cost in terms of cumulative weighted time-in-system (CWTIS). A minimum cost flow heuristic, incorporating a successive shortest path algorithm, was coupled with a critical arc schedule improvement heuristic. Our procedure iterated between these two heuristics to generate a cargo flow pattern and aircraft schedule. This research demonstrated the usefulness and efficiency of this heuristic in planning airlift for the channel cargo system. The FORTRAN programs which implement the heuristics are compatible with current AMC scheduling/advance planning tools. Given this compatibility, additional testing in conjunction with AMC's current planning tools (STORM, CARGPREP, and CARGOSIM) is warranted. Pending successful testing in this environment, implementation of these methods is recommended.

A HEURISTIC APPROACH TO DETERMINING CARGO FLOW AND SCHEDULING FOR AIR MOBILITY COMMAND'S CHANNEL CARGO SYSTEM

I. Introduction

I.1 General Issue

The United States Air Force's Air Mobility Command (AMC) is responsible for providing global logistical support to all U.S. military forces. This mission is accomplished through use of an extensive force that includes military aircraft, airbases, aircrews, maintenance facilities, and support personnel. Given the importance and resource expenditure of this mission, AMC maintains various organizations to ensure that routing and scheduling of its airlift resources is performed as effectively as possible. This research investigated a method for improving the routing and scheduling of these airlift resources.

I.2 Background

AMC's *channel cargo system* is a crucial logistical system for which routing and scheduling is planned on a monthly basis. A *channel* consists of a pair of origin and destination airbases, known as an *O-D pair*, between which AMC provides regularly scheduled airlift. Channels are established in response to various demands, such as the pickup and delivery of cargo or passengers.

Flexibility within the system allows for either direct connections between O-D pairs, where a cargo aircraft would fly directly from the origin airbase to the destination airbase, or may entail several intermediate stops before arriving at the destination. In addition, certain O-D pairs may not be serviced by a single route which connects the two bases. In this circumstance, *transshipping* is required, which occurs when cargo is downloaded from one

aircraft and uploaded onto another aircraft. Although this requires additional time and resources, it is an essential element of the channel cargo system and aids in increasing the utilization of aircraft and routes (Carter and Litko, undated:2).

The channels are classified as either *frequency channels* or *requirements channels*. Frequency channels, such as those missions typically flown to embassies, are scheduled at specified intervals and are not driven by cargo requirements. Requirements channels are routes flown between O-D pairs, with the number of missions flown based on the amount of cargo to be transported.

Given the importance of the channel cargo system's mission, a backup has been established in the event that AMC's assets cannot deliver all cargo as needed. This backup is the Civil Reserve Air Fleet (CRAF), consisting of civilian commercial transportation that is contracted as necessary by AMC's Tanker Airlift Control Center (TACC). The CRAF is an essential, significant source of augmentation for the channel system on an ongoing basis. In the past, expenditures on CRAF augmentation have exceeded expenditures on channel airlift (Shepherd, 1990:14). In dollar figures, the cost of augmenting AMC aircraft with commercial transport typically runs over $100 million annually (Ackley et al., 1991).

The advance planning for this channel cargo system is currently performed in a two-phase process which determines the number and types of missions flown, as well as a tentative schedule (see Figure 1). The primary goal of this advance planning is efficient use of channel system resources, including measures of timeliness and aircraft utilization (Robinson, 20 September 1993). In the past, AMC has accomplished this process on a monthly basis and as needed in response to special studies.

The first phase of the process uses a linear programming (LP) model, STORM (Strategic Transport Optimal Routing Model), to determine the number of missions needed. The goal of this LP is to select a mix of routes and aircraft to meet specified monthly cargo and frequency requirements while minimizing overall system cost. Overall system cost

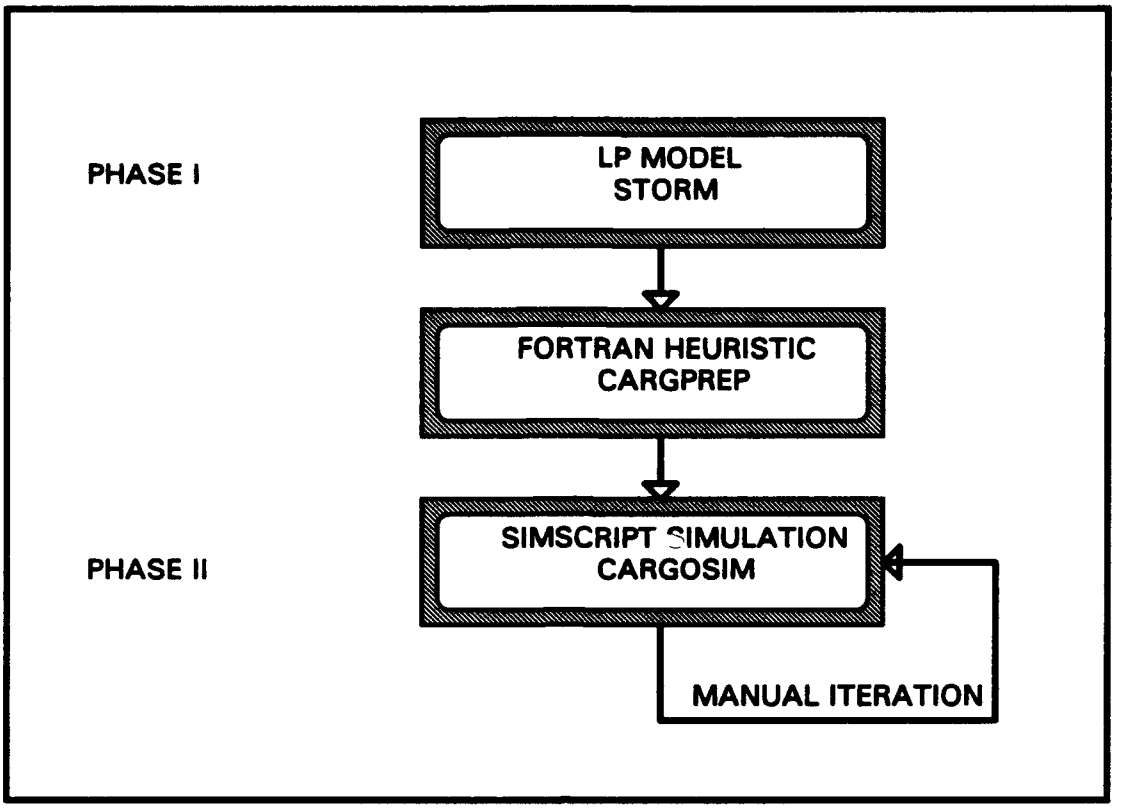


Figure 1. Current AMC Schedule Generation Process

includes military aircraft operations, cargo handling costs (e.g. transshipping), and commercial aircraft leasing (Ackley et al., undated:2). Limitations of the model include a restriction of at most one transshipment, lack of a time element in terms of delivery timeliness, and initial non-integer solutions (Whisman, undated:6-7).

The second phase employs a SIMSCRIPT II.5 discrete event simulation model, CARGOSIM, to validate the output from the STORM model. The objectives of the validation are to ensure the realism of the STORM output with respect to aircraft utilization and delivery timeliness (Del Rosario, 1993:4). The CARGOSIM model, in addition to requiring the STORM output of number of missions and aircraft, also requires a monthly flight schedule. This schedule is generated by a simple FORTRAN program called CARGPREP. CARGPREP evenly spaces identical missions generated by STORM throughout a given month (Robinson,

20 September 1993). For example, if a mission is to be flown four times during a month, CARGPREP would schedule successive missions exactly one week apart.

With the necessary inputs of the schedule, routes, cargo generation information, and aircraft properties, CARGOSIM simulates aircraft and cargo flow through the channel system and outputs measures of merit such as utilization, movement times, and port backlog (Robinson, 20 September 1993). The model factors in the element of *timeliness* of cargo delivery, which was lacking in the STORM model, in terms of average delay per cargo ton shipped between each O-D pair (Moul, 1992:1-5).

At this stage of the process, the Uniform Materiel Movement and Issue Priority System (UMMIPS) time standards are referenced. UMMIPS standards are described in the *DoD Materiel Management Regulation (DoD 4140.1-R):*

> Materiel shall be furnished to users on time, subject to constraints of resources and capability. The UMMIPS time standards shall be considered overall logistics system limits for the satisfaction of material requirements. Operational systems shall be designed to meet and, where economically feasible, to surpass the prescribed time standards (DoD 4140.1-R:5-19).

The UMMIPS time standards are in calendar days and vary according to origin and destination of the materiel. Within the channel cargo system, this allowable time delay typically varies between four and eight calendar days (Robinson, 20 September 1993).

The AMC analyst compares the simulation results to UMMIPS standards in order to modify the initial schedule from CARGPREP by changing the flight schedule (i.e. mission takeoff times) or to change the STORM output by varying the number of missions (Del Rosario, 1993:5). The modified schedule/mission set is reprocessed through CARGOSIM and compared to UMMIPS time standards again.

The analyst continues this manual iterative process of adjusting the schedule and checking its validity until UMMIPS standards are met (Rau, 1993:6). The process is involved and can take up to four days to complete. This two-phase process has also been used for other

applications, such as special studies of proposed modifications to the channel system (Del Rosario, 1993:6).

I.3 Previous AFIT Research

The Air Force Institute of Technology (AFIT) has conducted several research projects investigating ways to improve the AMC scheduling process: Moul (1992), Del Rosario (1993), and Rau (1993). The work of Del Rosario and Rau is most applicable to this research (all three efforts are discussed in more detail in Chapter II).

The research by Del Rosario was a mathematical programming approach to flowing cargo with a multi-period, multi-commodity network which modeled the channel cargo system (Del Rosario, 1993). The model was successful in flowing cargo within one of the geographic areas of the channel system for a week; however, the method had several limitations such as an "out-and-back" phenomenon and inability to model the large size of the channel system (Del Rosario, 1993: 75-78).

Rau's work was a mathematical programming approach to scheduling aircraft with a general job-shop model (Rau, 1993). The methodology was partially successful for a reduced size problem; however, it proved to be an inefficient use of linear programming techniques and failed to consider reflow of the cargo (Borsi, 23 July 1993).

The cargo flow model by Del Rosario was intended to be merged with the scheduling model by Rau. The limitations of the models prevented a successful merger.

I.4 Research Objective

The objective of this research was to develop an iterative process for scheduling airlift and flowing cargo that reduces cumulative weighted time-in-system (*CWTIS*) while maintaining or increasing cargo flow quantity. The objective was essentially the same as the thesis objectives of Del Rosario and Rau, who attempted to develop a two-step iterative process

originally proposed by Major John Borsi of the Air Force Institute of Technology (Borsi, 6 August 1992).

In this research, this process consisted of a *cargo flow heuristic* and a *schedule improvement heuristic*. The iterative nature of the process and the individual heuristics are discussed in later chapters. Figure 2 outlines the proposed modifications to AMC's scheduling process.

The objective of improving the schedule for the channel cargo system via reducing CWTIS will result in more efficient utilization of AMC airlift/personnel resources. This equates to saving the command money by allowing more cargo to be shipped on time by AMC assets and by transporting less commercially.

Additionally, a streamlined routing and scheduling methodology, compatible with the current scheduling process, could liberate AMC analysts from the current time-consuming three to four day process to evaluate the mission output by STORM and CARGPREP. Efficiency in advance planning and special studies will also be a benefit.

I.5 Problem Statement

Additional research is required to streamline the advance planning process for AMC's channel cargo system. *Given cargo/frequency of visit requirements, a fleet of aircraft, and possible routes, the goal is to develop, implement, and test an iterative, heuristic approach to effectively and efficiently schedule and load aircraft in order to increase the channel cargo system's efficiency by maximizing and maintaining cargo flow while reducing cumulative weighted time-in-system (CWTIS).*

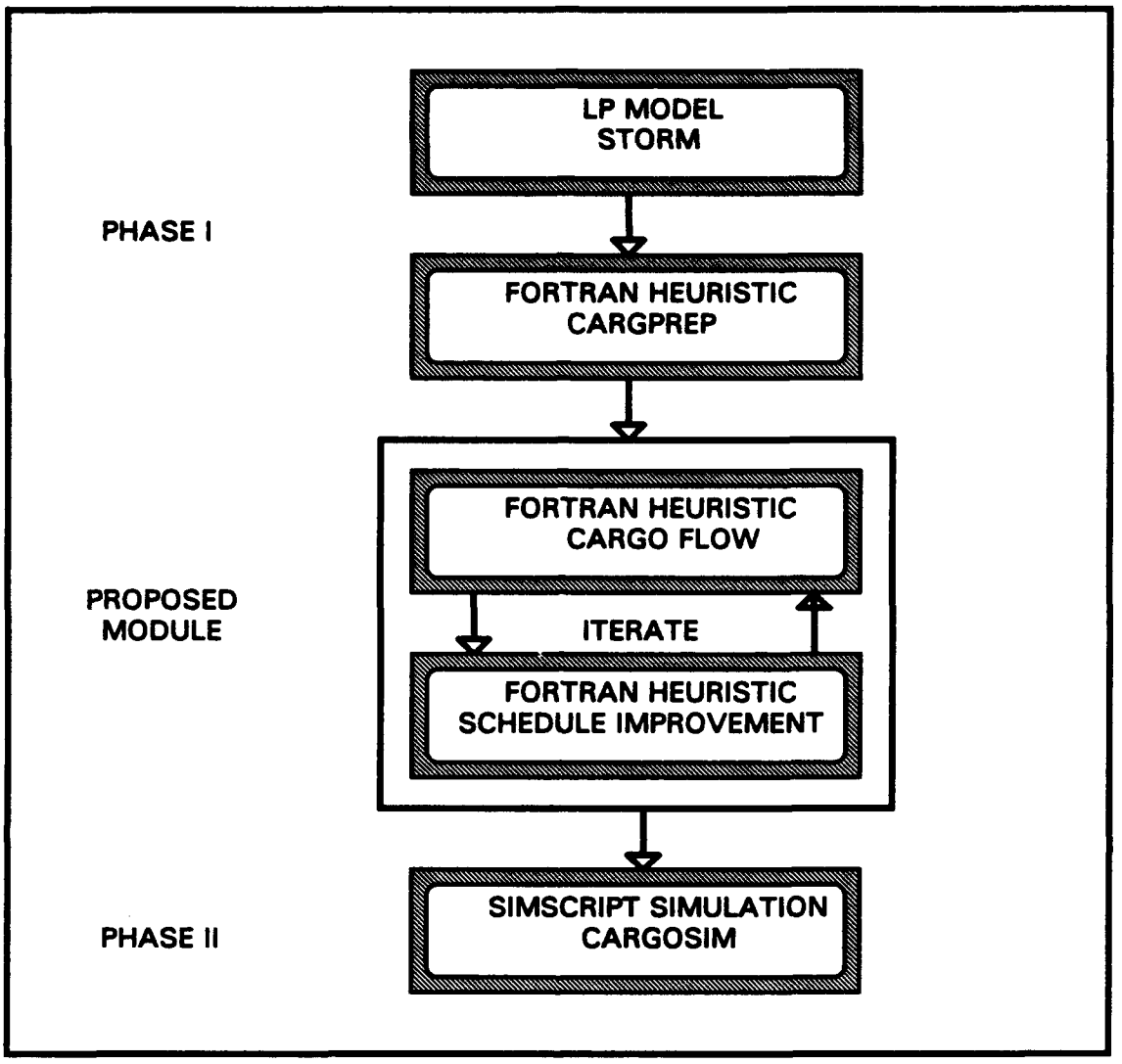


Figure 2. Proposed AMC Schedule Generation Process

I.6 Assumptions

Given this approach to modeling the channel cargo system, the following simplifying assumptions were used:

(1) All cargo requirements between origin-destination pairs are known with certainty.

(2) Cargo is classified by weight only and can be divided into an infinite number of subsets. Any mixture of cargo is allowed on a single aircraft. Other characteristics, such as

size and priority, are assumed to be the same for all cargo. Passenger requirements are not considered and do not affect aircraft cargo capacity.

(3) The number/type of aircraft available are known and remain constant.

(4) Each aircraft type has a specific cargo weight limitation. This weight limitation is not contingent on aircraft volume limitations. In accordance with previous AMC studies, the aircraft-specific weight limits were reduced appropriately to prevent violations of aircraft volume limits, as well as to realistically model loading efficiencies (Robinson, 20 September 1993). This weight limit reduction equated to using 1.5 tons per pallet instead of 2.3 tons per pallet in calculations.

(5) Airbases were assumed to be capable of handling an unlimited amount of aircraft and cargo and were assumed to be available 24 hours a day.

(6) Maximizing the cargo load of each aircraft was not considered. An aircraft did not need to be fully loaded before it could take off.

(7) Based on the results of Moul and Rau, CWTIS was considered an appropriate measure with respect to minimizing cumulative delay enroute (Moul, 1992; Rau, 1993). CWTIS is the cumulative sum of each cargo's weight multiplied by its time-in-system (TIS). TIS consists of all flight times plus any delay encountered enroute, including the delay encountered when cargo is at its origin airbase awaiting initial transportation.

II. Literature Review

II.1 Scope and Organization of the Review

Given the background concerning the channel cargo system and AMC's schedule generation process, this review briefly discusses some previous thesis efforts, including some of their shortcomings and limitations. These limitations provide a baseline for the alternative method discussed in Chapter III. Heuristics are then defined in relation to cargo flow and schedule improvement. Additionally, this review provides examples of heuristic applications in related routing and scheduling problems. A rudimentary knowledge of networks, maximum flow/minimum cost problems, and shortest path algorithms is assumed.

II.2 Previous Efforts

Graduate students at the Air Force Institute of Technology (AFIT) have conducted numerous research projects investigating ways to improve the AMC channel scheduling process: Moul (1992), Del Rosario (1993) and Rau (1993).

Moul produced a computer simulation for measuring cargo delay (Moul, 1992). The model he developed did not investigate rescheduling airlift or reflowing cargo.

The research by Del Rosario and Rau was directed toward applying mathematical programming techniques to the iterative procedure described in Chapter I (see Figure 2).

The research by Del Rosario was a mathematical programming approach to flowing cargo where a multi-period, multi-commodity network was used to model the channel cargo system (Del Rosario, 1993). The model was successful in flowing cargo within one of the geographic areas of the channel system for a week; however, the method had several limitations such as an "out-and-back" phenomenon and an inability to model the large size of the channel system (Del Rosario, 1993:75-78). This research uses the same data set as Del Rosario so that comparisons may be made between the efforts.

Rau used a mathematical programming approach to scheduling aircraft using a general job-shop model (Rau, 1993). The methodology was partially successful for a reduced size problem; however, it proved to be an inefficient use of linear programming techniques and failed to consider reflow of the cargo (Borsi, 23 July 1993).

The limitations of these models are evaluated in the following discussion.

II.3 Limitations of Prior Approaches

Applying mathematical programming to AMC's channel cargo system, Del Rosario and Rau discovered that the number of decision variables and constraints became so large that the computational capability of even the most state-of-the-art computer systems would be taxed. The attempt by Del Rosario to solve the problem as a multi-period, multi-commodity minimal cost flow formulation (M^2MCF) was unsuccessful primarily because of these computational limitations. According to Del Rosario, if AMC's entire channel cargo system were modeled as an M^2MCF, the estimated maximum number of variables in a one-month planning horizon is nearly 3 million and the estimated number of constraints is over 2 million (Del Rosario, 1993:38-39). This exceeds AMC's current computing capabilities, which can only solve linear programming problems with at most 160,000 variables and 20,000 constraints (Del Rosario, 1993:39). Del Rosario found that "because of the problem size and other modeling limitations discovered during [his] research, the presented M^2MCF model of the channel cargo system is currently not accurate enough to be useful as a scheduling tool" (Del Rosario, 1993:ix). Rau encountered similar problems in his attempt to reduce en route delays (Rau, 1993:41). Both efforts had to drastically partition the problem in order to reduce it to a workable size, and subsequently had to sacrifice optimality in the process.

In light of these previous efforts, given current model and computational limitations, optimal solutions for the channel cargo system may be nearly impossible to achieve using mathematical programming. From a practical research standpoint, a good feasible solution

may be satisfactory and far more tractable. As such, a heuristic approach to solving the problem has been adopted.

II.4 Heuristic Approaches

Heuristics are procedures that cannot guarantee an optimal solution. In fact, in some cases they cannot find even a feasible solution, although as Chapters IV and V show, for this research the problem has been defined so that a feasible solution can always be found. As defined by Borsi, "they are often based on insight into the fundamental nature of the problem and are used when an optimal procedure is unavailable or computationally intractable" (Borsi, 4 February 1994). As stated by Zanakis and Evans, they are "meant to provide good but not necessarily optimal solutions to difficult problems, easily and quickly" (Zanakis and Evans, 1981:84).

The following sections address some heuristics that apply to both cargo flow and scheduling.

II.4.1 Cargo Flow

The multi-commodity minimal cost flow problem (MMCF) was adopted in this research because it is a general way to model the flow of cargo between different O-D pairs. Chapter III describes the elements of this network problem in detail. Borsi recommended departing from the mathematical programming approach in the form of a successive, shortest path (S-P) heuristic, which is discussed in Chapter IV (Borsi, 15 July 1993).

In their discussion of optimization on networks, Syslo, Deo, and Kowalik describe the shortest path problem (finding the shortest path between two nodes in a network) as the most fundamental and also one of the most commonly encountered problems in the study of transportation and communication networks (Syslo et al., 1983:227). The *Busacker-Gowen Min-Cost Flow Algorithm (B-G)* is one method of solving a min-cost flow problem through repeated application of a shortest path heuristic.

Syslo, Deo, and Kowalik summarize an iteration of the B-G algorithm in two basic steps: (1) finding a shortest path in the network, and (2) modifying the network to account for the flow along this shortest path (Syslo et al., 1983:302). When a shortest path is found between the source and sink node, flow is sent along that path until the path reaches saturation or the total flow reaches the target value for the source. If no shortest path can be found, the B-G algorithm terminates for the current source/sink combination.

If the current path is saturated, the *network must be modified* to account for this flow. This modified network has the same structure as the original, except that certain costs and capacities will change. The capacities along the arcs of the shortest path will be decremented according to the flow established. If any of these arcs is saturated, the capacity along this arc is set to zero and the respective cost is set to infinity. Additionally, all arcs with nonzero flow have "fictitious" arcs introduced in the reverse direction. These reverse arcs have capacity equal to the current flow through the forward arc and cost equal to the negative of the cost of the forward arc. These reverse arcs allow flow reduction in subsequent iterations of the B-G algorithm.

For any flow level achieved, the B-G algorithm produces the minimum cost solution. By specifying an infinite target flow, the B-G algorithm will solve a max-flow problem as well; however, it is considered an inefficient method of solving this problem (Syslo et al., 1983:306). The multi-commodity nature of the channel cargo system could be added to the B-G algorithm through additional iterations with updated source and sink nodes. Some of these considerations will be discussed in Chapter IV. There are a variety of S-P algorithms, including algorithms by Bellman, Dijkstra, Dantzig, Whiting and Hillier, and Floyd (Bazaraa et al., 1990:625). Chapter IV discusses the selection of the specific S-P algorithm in more detail.

II.4.2 Schedule Improvement Approaches

There are over 100 heuristics reported in the literature for scheduling. According to Zanakis and others, of those, there are 19 that deal specifically with *improvement* (Zanakis et al., 1989:93). Improvement heuristics typically begin with a feasible solution and successively improve it by a sequence of local exchanges. The goal is to maintain a feasible solution throughout the procedure (Zanakis et al., 1989:89).

The heuristic developed in this research is similar to the *k-opt* improvement heuristic used to solve traveling salesman problems. K-opt is an improvement heuristic which "...affects interchanges between the components of [the] schedule to improve costs" (Bodin, 1983:134). The heuristic developed in this research seeks to improve costs by interchanging existing arcs in the flow paths with other, potentially new ones. Chapter V discusses the precise methodology employed.

III. Overall Methodology

III.1 General

As discussed in Chapter I, the goal of this research was to develop an iterative procedure for scheduling airlift and flowing AMC's channel cargo. The approach consists of two major components, a flow algorithm and a schedule improvement algorithm, which are discussed in Chapters IV and V, respectively. This chapter discusses precisely how the components interact with each other to achieve an improved cargo *flow pattern*, which is defined as a set of feasible arc flows in a directed network. The chapter also discusses this research's iterative improvement approach, the network formulation, and methods employed to reduce problem size and complexity.

III.2 Integration

The goal of the iterative method developed in this thesis is to maximize the tonnage of cargo delivered while minimizing the time commodities spend in the channel system. The method, suggested in 1992 by Borsi and explored by Del Rosario and Rau (Del Rosario's research addressed Step One while Rau's addressed Step Two), was described by Rau as

> a two-step, iterative process. In Step One, given any schedule, a flow of cargo is determined based on this schedule. The cargo is categorized by its quantity (weight) and its type (origin and destination). Step One determines the quantity and type of cargo that is loaded onto or taken off each aircraft as the cargo is transported from one airbase to another on its assigned path. Step Two modifies the flight departure times and revises the overall schedule based on this cargo flow. Returning to Step One with the revised schedule, the cargo flow is modified based on the new flight times (Rau, 1993:7-8).

Figure 3 demonstrates how the iterative improvement algorithm integrates the two steps. It is discussed in more detail later. The flow of cargo is dependent upon the existing network, which itself is created by a schedule. Any change to the schedule will necessarily alter the structure of the network and offer the potential that an improved flow pattern can be found on the altered network.

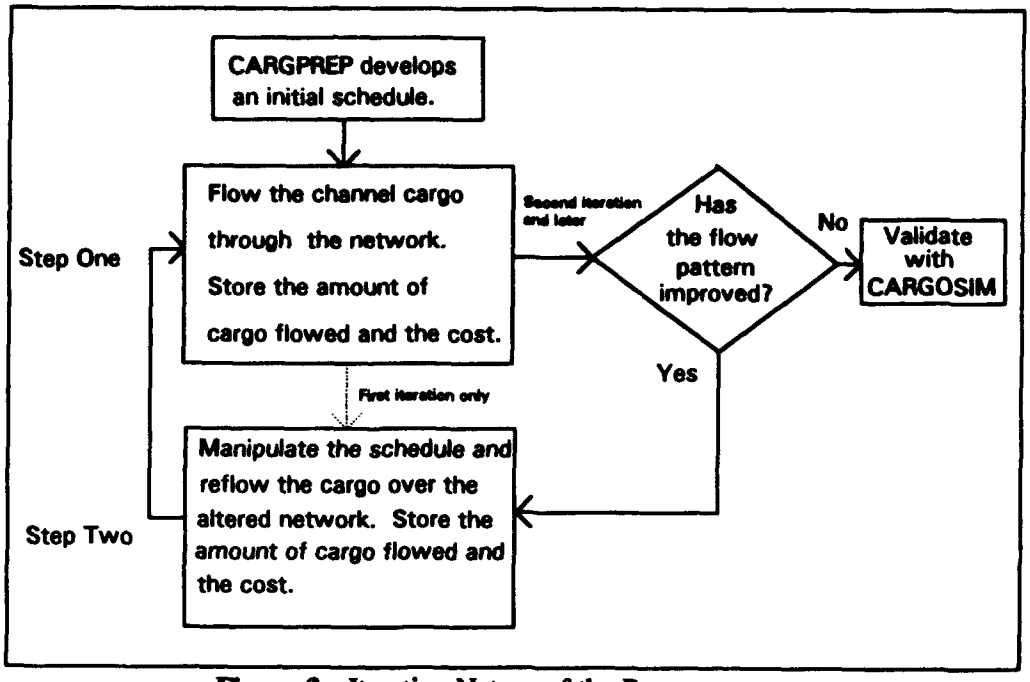


Figure 3. Iterative Nature of the Process

Step One, flowing the cargo, is addressed in Chapter IV, and Step Two, improving the schedule, is discussed in detail in Chapter V. While Steps One and Two rely on each other, the determination of whether the process has actually led to improvement combines output from both. Recall that the process is designed to iteratively approach, or converge on, a good feasible solution. The algorithm terminates when the process ceases to improve the flow pattern of the cargo.

Each commodity that is flowed has its own individual cost, called *weighted time-in-system* (WTIS), measured in day-tons. WTIS is the product of a commodity's tonnage and its time-in-system. The cumulative WTIS (CWTIS) of the entire flow pattern is the sum of each commodity's WTIS and is the measure used to assess flow improvement. Note that this measure places greater significance on larger shipments since, if a large shipment is delayed, its impact will be felt more sharply than if a smaller shipment is delayed by the same amount.

A complete iteration through the iterative improvement algorithm consists of a pass through both Step One and Step Two. Step One produces a flow pattern with associated CWTIS;

Step Two then attempts to alter the schedule in order to produce a flow pattern with a CWTIS equal to or less than that produced by Step One. The CWTIS produced by Step Two is compared to the CWTIS produced by Step One on the next iteration to determine if the algorithm should proceed or terminate. As Figure 3 demonstrates, after Step One of the first iteration there is no previous value of CWTIS to compare with to assess improvement. The algorithm proceeds directly to Step Two. On subsequent iterations, however, the algorithm assesses improvement before proceeding to Step Two.

As Figure 3 shows, when the algorithm determines that the flow pattern has not improved, the solution is passed to CARGOSIM for validation.

III.2.1 Flow Pattern Improvement

The assessment of iterative improvement is described below. The following cases discuss flow pattern improvement:

Case 1: If the amount of cargo flowed in Step One is the same as that flowed in the previous iteration, the flow pattern improves if CWTIS decreases.

Case 2: A flow pattern is also improved if, after a pass through Step One, *more* cargo was flowed than on the previous iteration. In essence, the changes to the schedule made in Step Two changed the network sufficiently to allow more cargo flow. Note that any pattern that flows more cargo is considered superior, regardless of CWTIS.

III.2.2 Lack of Flow Pattern Improvement

The following cases demonstrate instances where the iterative improvement algorithm would terminate due to a lack of improvement.

Case 1: If the amount of cargo flowed remained constant relative to the previous iteration, but CWTIS did not decrease, the flow pattern has *not* improved. Since the algorithm continues to iterate only when improvement occurs, it terminates after Step One. If CWTIS remained constant, the flow pattern from Step One and the pattern from the previous pass through Step Two both

represent acceptable feasible solutions. To determine which is preferred, the analyst can examine the time-in-system distribution and/or the transshipment distribution of the flow patterns, both of which are provided as output.

Case 2: Ideally, after a schedule has been improved by Step Two, the next pass through Step One should be able to flow at least as much cargo as on the previous iteration. It can do so by using the flow pattern generated by Step Two. However, because the cargo flow algorithm is a heuristic, a subsequent application of Step One may generate a flow pattern that delivers less cargo. Thus, the flow pattern has not improved. The algorithm terminates and stores the most recent pattern found by Step Two as the solution. The flow pattern found by the final pass through Step Two will represent the best solution in this case because it flows a greater amount of cargo than the final pass through Step One.

At the end of the iterative process described here, the AMC analyst possesses a flow pattern and an actual flight schedule covering the planning horizon. The flow pattern lists each piece of flowed cargo and the path it used, showing mission numbers, node numbers, bases, and associated times. The flight schedule is in the form of a list of the nodes in the network and the times associated with them. This list can easily be adapted for validation in CARGOSIM.

III.3 The Network

A specialized network was used to model the AMC channel cargo system. The network is defined by the cargo requirement and the current schedule. The initial schedule was produced using output, shown in Appendices D-G, from AMC's STORM and CARGPREP models. Subsequent schedules were produced by the schedule improvement algorithm.

The nodes of the network represent three different entities: 1) a cargo generation node (source node), 2) a mission airbase node where an individual sortie makes a scheduled stop, and 3) a commodity-dependent destination node (sink node). All source and mission airbase nodes have times associated with them. For a source node, the time represents the availability time for a

commodity. For example, a time of 3.5 for a source node means that a commodity is delivered to its origin airbase at the 3.5 day point within the planning horizon. For a mission airbase node, the associated time represents the time that the given aircraft is at the specified airbase. Furthermore, every node's associated base is identified by its ICAO (International Civilian Aviation Organization), a distinct four-character designator. The source node has two associated ICAOs, one for cargo origin and one for cargo destination. Each distinct cargo destination has a sink node with matching ICAO.

As modeled, the channel system network is a *directed graph* with the arcs only directed forward in time. That is, an arc originating at a node can only terminate at a node with an equal or later associated time. This models the time dependency of the problem. *The cost of each arc represents the time spent performing its associated activity.* The arcs can represent five different activities: 1) a delay encountered when cargo is at its origin airbase awaiting initial transportation, 2) the time it takes a given aircraft to fly a leg of its route between two airbases (nodes), 3) ground time for an aircraft along its route (which implies either an aircraft-specific standard ground time or a remain-over-night (RON) time), 4) transshipment, which represents time spent at an airbase while offloading cargo from one aircraft and reloading it onto another, and 5) arrival at the final destination (these arcs are zero-cost sink connection arcs which connect a mission airbase to an applicable sink node of the same ICAO).

Figures 4 through 8 illustrate an example network for the channel cargo system. In the figures, each horizontal sequence of nodes represents a single aircraft flying a single mission. Figure 4 illustrates the three node types used in the network, as well as the mission leg arcs and mission ground time arcs. Figure 5 includes the time and ICAO information associated with the node set (for clarity, the ICAOs have been shortened to a one-letter designation). Figure 6 depicts the arcs that connect source node number one to the mission airbases of proper ICAO and time.

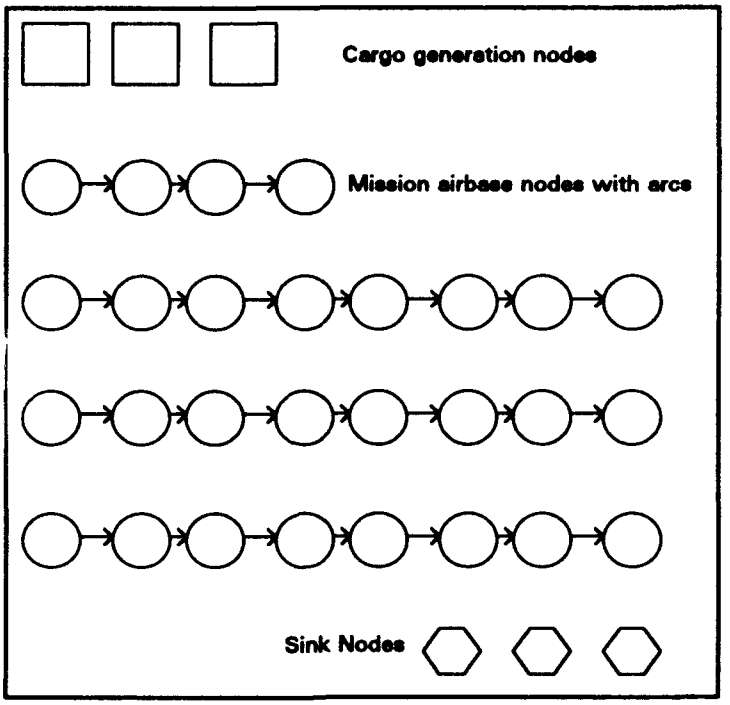


Figure 4. Three Types of Nodes in the Network

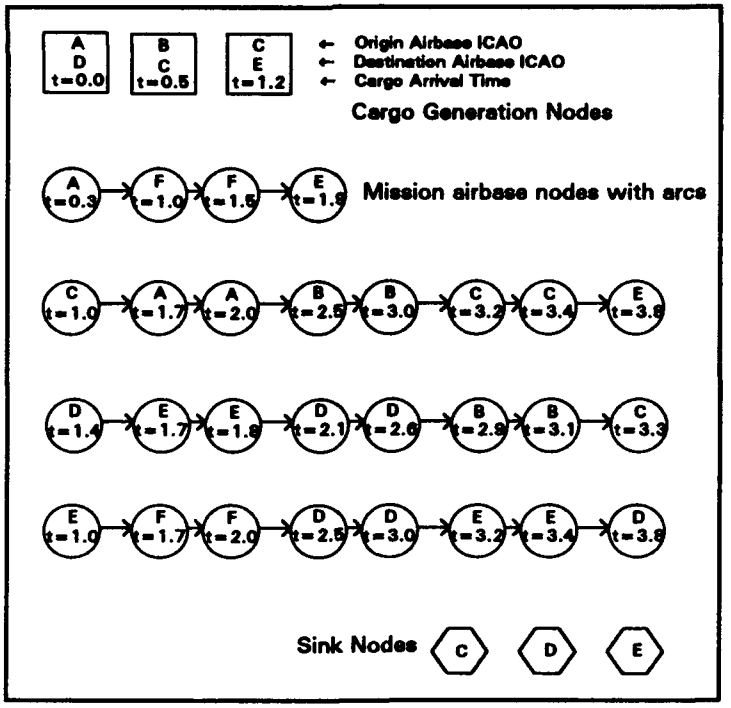


Figure 5. Times and ICAOs Associated with the Network

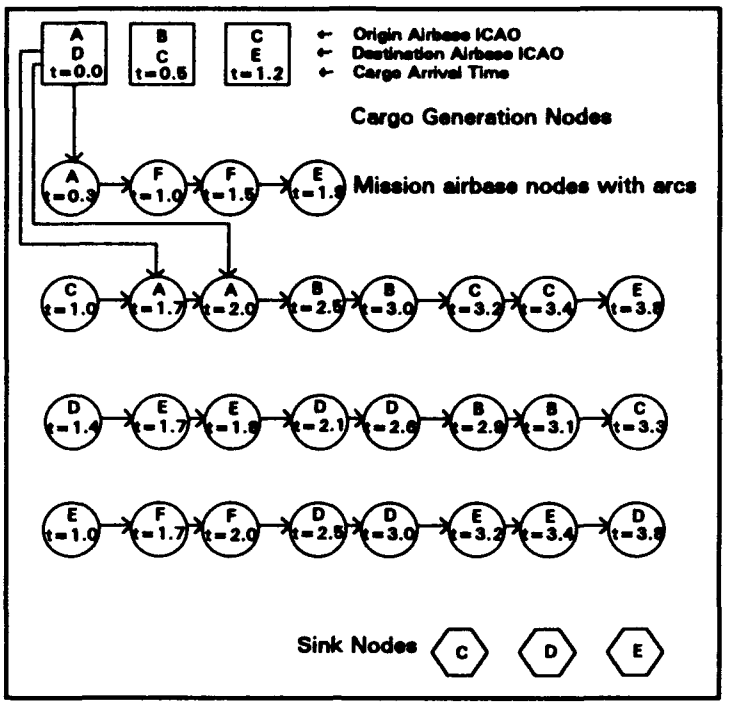


Figure 6. Arcs Connecting Source Node 1 with Base 'A'

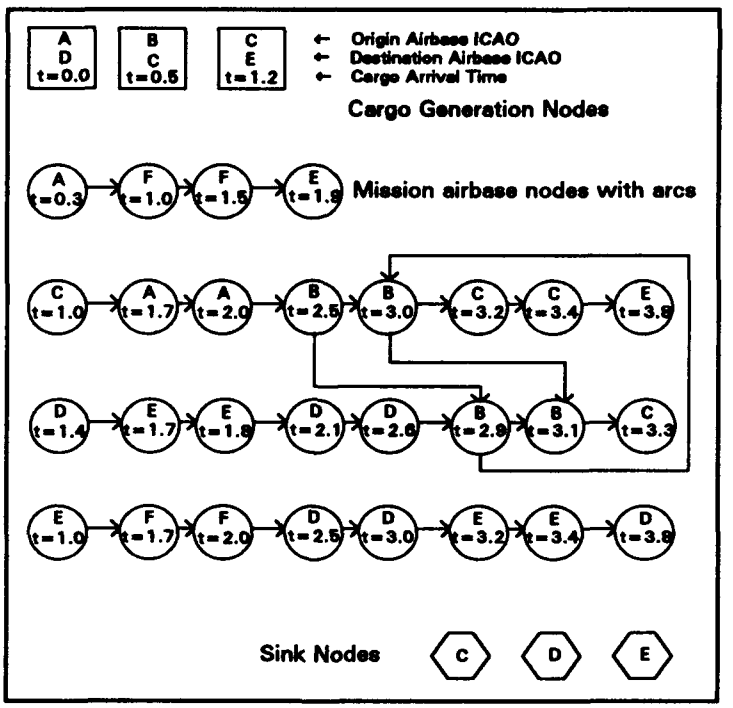


Figure 7. Transshipment Arcs for Airbase 'B'

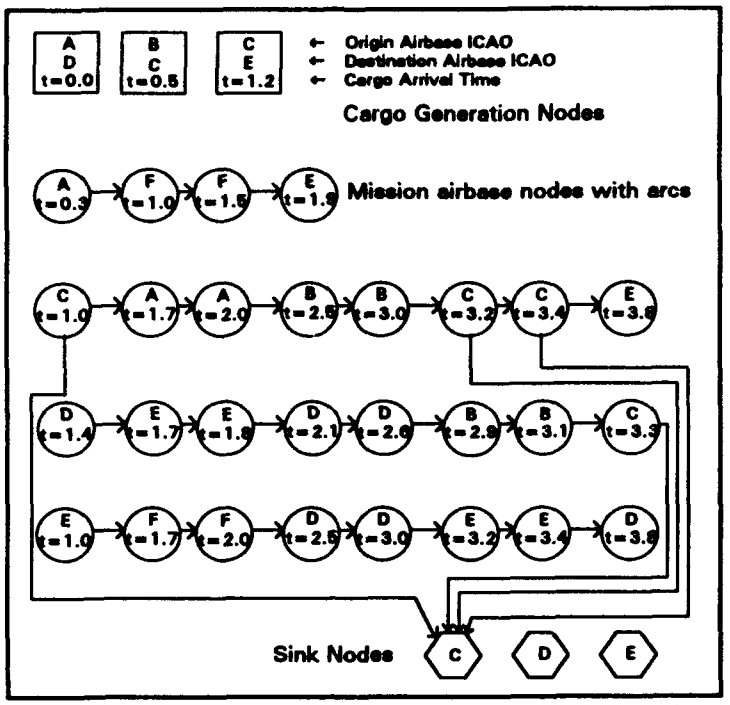


Figure 8. Zero-cost Arcs Connecting Airbase 'C' with its Sink Node

Figure 7 depicts the arcs connecting airbases of the same ICAO (only "B" in the figure) to allow for cargo transshipments. Figure 8 illustrates the zero-cost arcs connecting mission airbases to a sink of the same ICAO. A sink node of a given ICAO is generated only if a cargo node exists with that destination ICAO.

III.4 Reducing the Problem Size

Previous efforts were unsuccessful in modeling the entire channel cargo system due to its large size and complexity. Del Rosario lists some typical data for the channel system for an average month based on his discussions with AMC personnel (Del Rosario, 1993:38):

Number of commodities: 437

Number of sorties: 528

Average number of legs per mission: 3

Assuming that each commodity has seven nodes to represent a week's worth of cargo arriving on consecutive days, 3059 source nodes would be required in a network representing a

week's activities. A sortie with three legs would have six mission airbase nodes, so the total number of mission airbase nodes is 3168. Assuming some duplication in commodity destination, the total number of sink nodes is assumed to be 200. Therefore, the total number of nodes required to model a week's activities is approximately

Source nodes: 3059

Mission airbase nodes: 3168

Sink nodes: 200

Total: 6427 nodes in the current network formulation.

This is a sizable network, so storage limitations must be considered. Storage of the arc set as an NxN weight matrix would require $6427^2 = 41,306,329$ bytes. Additionally, the algorithms used in this thesis effort require storage of costs, capacities, and flows for each arc, as well as temporary values for these arc parameters. This would require six of these NxN matrices, or approximately 250 million bytes. This is an inordinate amount of storage capability (in Random Access Memory) and would tax AMC's computational capabilities.

In response to this huge memory requirement, a modified *linked adjacency list* was developed. A linked adjacency list groups arcs according to their node of origin, which is known as a *forward star* (Syslo et al., 1983:225). Instead of storing a large number of irrelevant (i.e. zero or infinity) values in an NxN matrix, the adjacency list only contains data for arcs that exist in the network formulation. The benefits of this data representation are more evident in a sparse network and diminish as the number of arcs increases.

In addition to this data representation, the total size of the arc list was reduced. If an arc is established between any two mission airbases of the same ICAO, directed according to time precedence, there is a potential for (m)(m-1)/2 arcs for each ICAO airbase within the mission airbase subgraph (where m is the number of same-ICAO airbases for any ICAO present). This can result in a very large arc set. For the Europe/Southwest Asia (E/SWA) sub-problem addressed in

this thesis, our original implementation produced approximately 180,000 arcs for a 2,293 node network.

This proliferation of arcs was prevented by establishing a *spanning path* for mission airbase nodes linked in a time sequence. Figure 7 displays such a path for the "B" airbase nodes. If two nodes with the same ICAO have the same time, arcs are added to the path which allow *cycles* between the nodes representing the airbases. By implementing the spanning path, the arc set is reduced to length (m-1), assuming no cycles within the path. This significantly reduced the number of arcs in the E/SWA problem to approximately 20,000.

III.5 User-preferences

Within the program are several problem-dependent parameters which the user may change to enhance program performance and realism. Specifically, these parameters are 1) the number of transshipments allowed on a particular path, 2) the policy used when transshipping, e.g. restricting transshipments to certain airbases, 3) the criteria used to sort the cargo, 4) the order in which the missions are examined by the schedule improvement algorithm, 5) the maximum number of passes per iteration through the schedule improvement algorithm, 6) the maximum number of alternate paths examined by the shortest path algorithm, and 7) the maximum number of iterations of the program. Parameters 1 through 6 are discussed in Chapters IV and V, as appropriate. Parameter 7 is used as a ceiling on running time. The possible values for these parameters are shown in Appendix J.

The algorithm developed in this thesis is meant to be a tool for better planning and is designed to be as dynamic and flexible as possible. It provides the analyst the opportunity to tailor the process to the constraints of the problem.

IV. Flow Methodology

IV.1 General

This chapter discusses Step One of the iterative improvement algorithm outlined in Chapter III. Given a mission schedule and cargo requirements, the network is developed. Then the problem is to minimize the CWTIS incurred while flowing cargo from source nodes to sink nodes. This is a multi-commodity minimal cost flow (MMCF) problem, where each commodity represents a source-sink combination.

There are a variety of approaches to solving the MMCF, several of which were outlined in Chapter II. These methods vary considerably in terms of tractability and speed. The concepts of the *Busacker-Gowen Min-Cost Flow Algorithm (B-G) were modified into a cargo flow heuristic* for this research. The modifications were motivated by the objectives of this research.

IV.2 Motivation for the Flow Heuristic Approach

The cargo flow heuristic had to exhibit two characteristics in this research: 1) it had to be computationally efficient, and 2) it had to allow effective control of the flow paths. Modifications to the B-G algorithm were undertaken with these two characteristics in mind.

As noted in Chapter II, the B-G algorithm introduces "fictitious" reverse arcs into the network in response to flow. These reverse arcs allow flow redirection in subsequent iterations, allowing the algorithm to develop a min-cost solution. Within our channel system network, these reverse arcs were detrimental to an efficient implementation of the flow heuristic.

Chapter III outlined the linked adjacency list used to store all network arc data. If reverse arcs were introduced for each flow path, the list would need to be constantly updated in order to insert these arcs. In addition to reducing the computational speed of the flow

heuristic, these arc additions would increase memory requirements and reduce the maximum size problem the iterative improvement algorithm could process.

Path control would also be degraded with the addition of the reverse arcs. The quality of a flow path involved several factors besides time-in-system, including number of transshipments and flow capacity. Reverse arcs could reverse this flow on subsequent iterations, favoring paths that are not desirable when considering all factors.

Unfortunately, the mechanism that allows the B-G algorithm to derive the min-cost solution, addition of reverse arcs, is detrimental to both cargo flow characteristics and was not implemented. With this modification justified, the cargo flow heuristic can be outlined.

IV.3 Description of the Cargo Flow Heuristic

The cargo flow heuristic followed the same two basic steps of the B-G algorithm: (1) *finding a shortest path* in the network, and (2) *modifying the network* to account for the flow along this shortest path. Figure 9 outlines the steps of the cargo flow heuristic. The heuristic begins by selecting the current commodity, its quantity, and the respective source and sink nodes. Once this is performed, Step One is implemented.

Step One consists of a call to the shortest path algorithm. If a shortest path is found between the source node and sink node, the algorithm proceeds to Step Two. If no path can be found, the heuristic repeats Step One for the next commodity. If no more commodities remain, the cargo flow heuristic terminates.

Step Two modifies the network to account for flow along the shortest path. This modified network has the same structure as the original, except that certain costs and capacities will change. The flow quantity for the path is the minimum of the capacities of the constituent arcs and the commodity quantity remaining. The capacities along all the arcs are

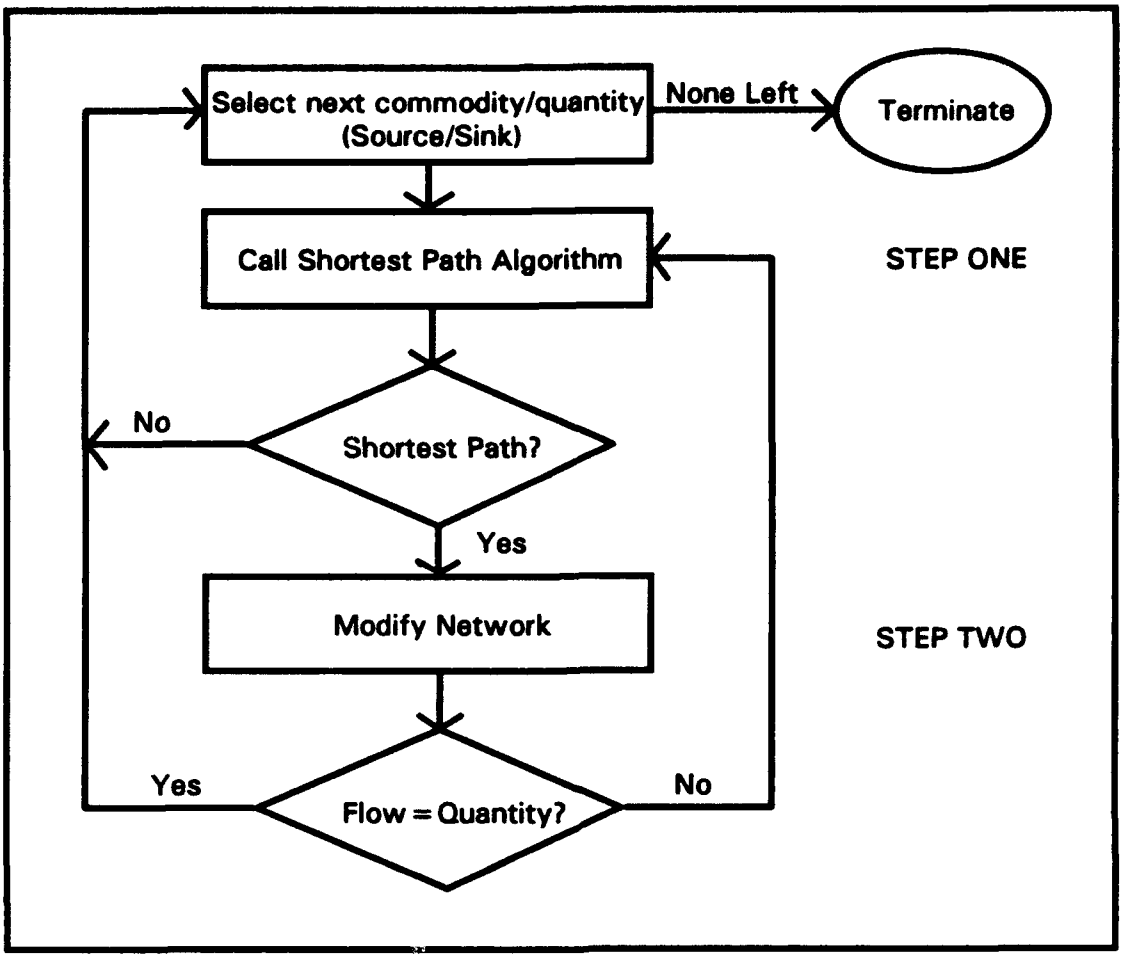


Figure 9. Steps of the Cargo Flow Heuristic

decremented according to the flow established. If any of these arcs is saturated, the capacity along this arc is set to zero and the respective cost is set to infinity.

The cargo flow heuristic deviates from the B-G algorithm at this point. The B-G algorithm would also introduce the "fictitious" reverse arcs when modifying the network. As discussed, these arcs hinder the performance of the flow heuristic and are not introduced.

Once the network has been modified, the heuristic compares the total flow for the current commodity (the sum of all individual path flows) to the commodity quantity. If the flow equals the commodity quantity, the algorithm selects the next commodity. If the flow is less than the commodity quantity, Step One is repeated.

The processing sequence of the commodities is non-trivial and can affect the overall system CWTIS. Obviously, as the network arcs become infeasible due to flow of subsequent commodities, options for a shortest path may be degraded. Some considerations in commodity sequencing and path selection are addressed in Chapter VI.

IV.4 Shortest Path Algorithm

The selection of a specific shortest path (S-P) algorithm is important in terms of computational efficiency. As the S-P algorithm is the building block for the cargo flow heuristic and is called many times, an efficient algorithm was desired. Dijkstra's S-P algorithm was selected due to its efficiency and ease of coding.

IV.5 Alternate Path Selection

Often, the initial path found by Dijkstra's algorithm is not the most preferable in terms of transshipments and flow capacity (early analysis results on the E/SWA problem indicated this). Given a shortest path connecting a source node to its respective sink node, there is a definite possibility that an alternate shortest path could exist within the network.

The *transshipment* issue complicates the shortest path computations considerably. Transshipping cargo entails downloading it from an aircraft and uploading it to another aircraft. Obviously, this takes time and resources and should be avoided unless absolutely necessary. AMC lists the current cost of transshipping as approximately $176 per ton (Robinson, 20 September 1993). Additionally, discussions with AMC analysts indicate that both the STORM and CARGOSIM tools assume a *maximum of one transshipment per commodity* (Robinson, 20 September 1993). This one transshipment constraint, as well as possible restrictions on airbases that allow transshipments, predicates emphasis on proper path selection.

The nature of Dijkstra's S-P algorithm allows construction of alternate paths once the initial path is found. Within the algorithm, nodes are given *permanent labels* when the

shortest distance from the input source node to the specified node is found. This labeling process is accomplished in a non-decreasing fashion, as the nodes that have the shortest path lengths relative to the source node will be found and labeled before other nodes in the network.

This fact was exploited in attempting to construct the alternate paths. Only the nodes that had been permanently labeled were candidates for an alternate path of the same length (time-in-system). All other unlabeled nodes were not close enough to the source to constitute a path of the same length.

Once the initial shortest path was found, the *predecessor array* was instrumental in determining alternate routing through the network. The predecessor array is an array which allows one to trace a path from the sink node back to the source node. This node set was scanned, starting at the node prior to the sink. At each node of the scan, the arc list was reviewed to see if any arc, originating at a different permanently labeled node, ended at the current node. If such an arc existed, an alternate path was found.

Having found an alternate path, its number of transshipments, flow capacity, and actual time-in-air were determined. These characteristics were compared to the initial path characteristics and the superior path was stored. With the other characteristics equal, a path is superior if it has fewer transshipments. If two paths have the same number of transshipments, the path with higher flow capacity is selected. This process continues for each alternate path found.

Figures 10 and 11 demonstrate the nature of the alternate path logic. Figure 10 displays an initial path found by the S-P algorithm. This path is not desirable, as it has one transshipment and also places the cargo on two out-and-backs (returning to the same airbase before final delivery) prior to final delivery at base E. Figure 11 displays the path that would be found and selected using the alternate path routine incorporated into Dijkstra's S-P algorithm. This path has no transshipments and no out-and-backs. As the routine traced back

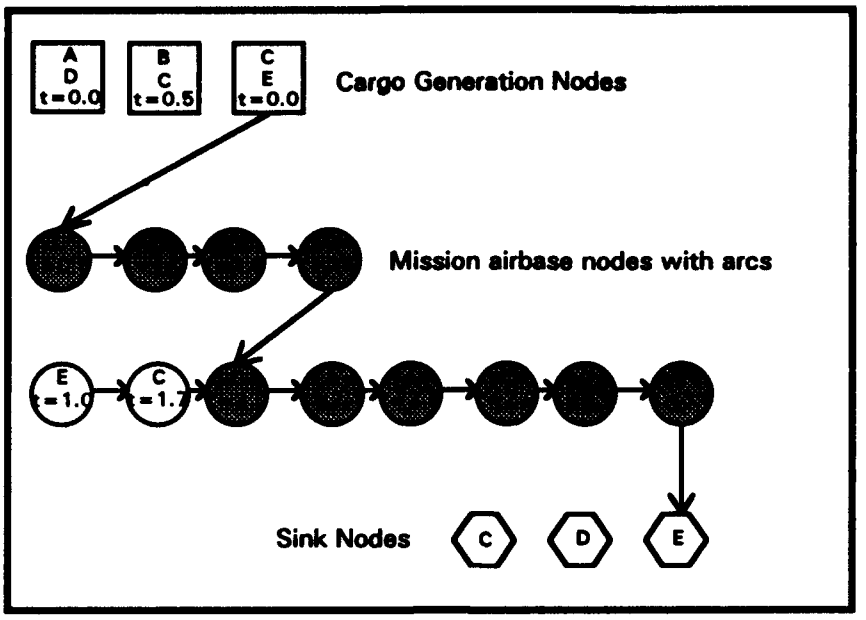


Figure 10. An Initial Path Found

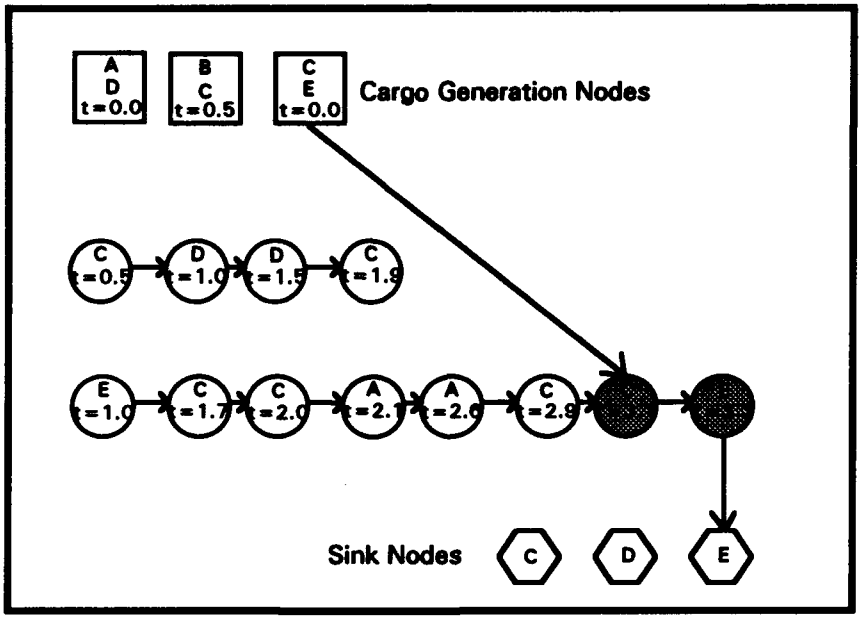


Figure 11. A Superior Alternate Path

from the node prior to the sink (E at t=3.3) in the initial path, it would proceed to the C node at t=3.1. The review of the arc list at this point would indicate that the source node, which has been permanently labeled, has an arc going into that C node. This alternate path is found and analyzed as superior to the initial path found. It is stored and the search continues.

IV.6 Path Compression

The nature of the network, as well as the implementation of the alternate path algorithm, produced some paths with a large number of nodes. As discussed in Chapter III, a spanning path was designed to allow transshipments while minimizing the number of arcs required in the network. Unfortunately, these spanning paths could result in some flow paths of inordinate lengths, with meaningless intermediate nodes.

A compression algorithm was developed to shorten all flow paths to only the essential nodes (reference the main program listing in Appendix B). This algorithm scanned the predecessor array returned from the shortest path algorithm, looking for nodes with the same ICAO separated by intermediate nodes. If this condition occurred, the intermediate nodes were eliminated from the path. The arc list was then scanned to determine if an arc existed to flow between these two nodes. If an arc was present, its flow quantity was modified. If an arc was not present, it was created and added to the arc list with an appropriate flow value.

IV.7 Analysis of the Out-and-Back Phenomenon

An ***out-and-back (O&B)*** occurs when cargo travels on mission legs that form a cycle, returning the cargo to the same airbase it has visited previously, prior to final delivery at the destination airbase (see Figure 12). This is an undesirable condition because it reduces aircraft flow capacity along those legs and would increase fuel costs. It is preferable, in *most* cases, to download the cargo and upload it at a later time, circumventing the out-and-back (Robinson, 1 February 1994). Unfortunately, this will often increase the number of transshipments that the cargo must undertake.

As mentioned above, *most* of the time O&Bs should be avoided; however, there are competing objectives: the cost of preventing the out-and-back by transshipping (downloading and uploading later to the same mission) versus the cost of leaving the cargo on the aircraft and incurring the reduced flow capacities/increased fuel costs along those mission legs. In certain cases, where the O&B is short and the cargo is either difficult or impossible to download at the airbase (i.e. position within the aircraft and airbase equipment restrictions), an O&B might be preferable. This *could* be modeled in this methodology and given a user parameter to specify which condition is preferable. In terms of the *objective of this research,* maximizing flow while minimizing CWTIS, *the option of transshipping the cargo to prevent the out-and-back is always considered preferable* because it increases future flow capacity along the intermediate mission legs without increasing the CWTIS. The compression algorithm, originally designed to eliminate the transshipment spanning path in the paths returned from Dijkstra, was easily modified to eliminate the out-and-backs as well.

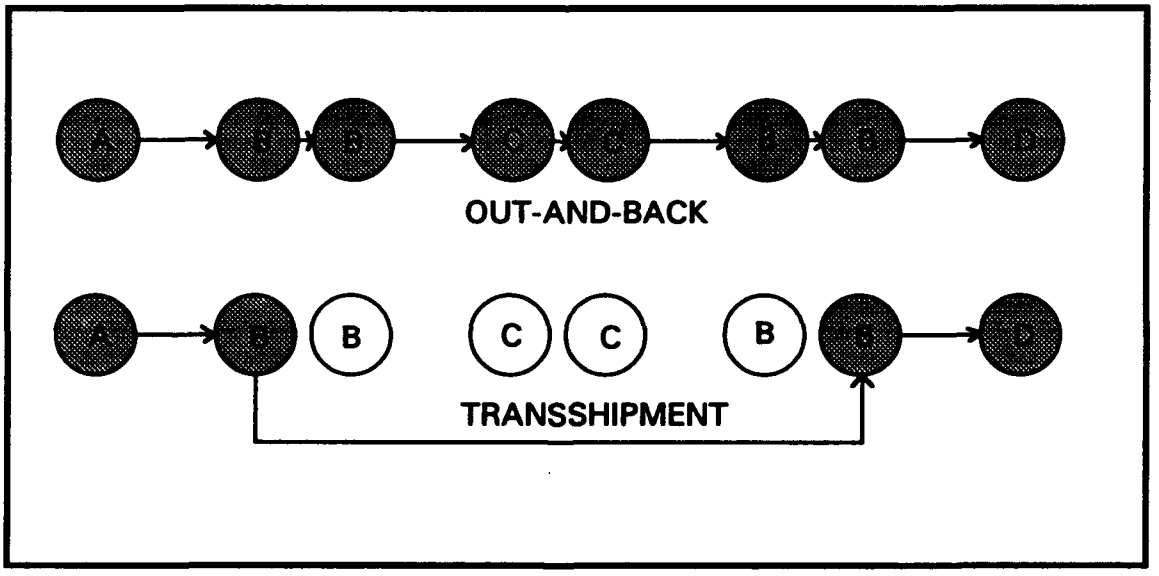


Figure 12. Prevention of the Out-and-Back Phenomenon

V. Schedule Improvement

V.1 General

This chapter addresses Step Two of the iterative improvement algorithm (IIA) discussed in Chapter III. Step Two, hereafter referred to as *schedule improvement*, involves shifting mission start times in the hope of reducing the CWTIS of the flow pattern developed by Step One, as discussed in Chapter IV. The method developed here is similar to Rau's method only in its intent. Rau modeled AMC's channel cargo system as a *job-shop scheduling problem* (where a machine corresponded to an aircraft flying a single flight leg and a job was a requirement to transport cargo from one airbase to another) and employed the concept of *semi-active time tabling* to develop a new schedule (Rau, 1993:21). While his method was certainly an adequate approach to schedule improvement, it did have a limitation: its scope was too narrow.

This limitation was a direct result of his use of semi-active time tabling, which "produces a schedule in which no operation could be started earlier without altering the processing sequence" (Rau, 1993:18). In other words, Rau only perturbed the schedule to the point that all existing arcs in the network remained feasible and all cargo flowed along the same path it used prior to schedule alteration. The approach developed here goes further by reflowing cargo along a different path if doing so improves the flow pattern.

Given a feasible solution, the general approach for schedule improvement involves manipulating the schedule to a degree that preserves the feasibility of the solution and reduces its overall cost. For this research, a feasible solution is one in which the amount of cargo flowed between each O-D pair is not reduced.

Recall that the schedule consists of a sequence of missions with scheduled start times. The approach adopted in this research was to *reschedule* the missions by adjusting their start times in an effort to deliver cargo to the customer in a more timely fashion. Obviously, it is

possible to reschedule a mission to an earlier or a later time; the method developed here only addresses shifting a mission to an earlier time. Shifting to a later time is left to future research.

Determining the amount by which a mission is rescheduled is not a trivial matter. The existence of transshipment arcs within the network complicates this issue. Shifting the mission start times may have a considerable impact on the transshipment arcs, i.e. many feasible arcs may become infeasible. An arc is feasible when its direction is forward in time and becomes infeasible when its direction changes to backward in time. That is, for an infeasible arc the time associated with its head node is earlier than the time associated with its tail node. As mentioned in Chapter III, arcs represent various physical activities, so they cannot go backward in time. The flow of cargo may be affected as the transshipment arcs are affected.

The heuristic contains a series of checks to maintain the feasibility of the solution at all times. This involves determining when improvement occurs and when it does not. Specifically, these checks must ensure that no previously-flowed cargo goes undelivered.

V.2 The Schedule Improvement Algorithm

At Step Two of each iteration of the IIA, the schedule improvement algorithm is applied sequentially to every utilized requirements mission in the existing schedule and then generates a new schedule. The new schedule in turn generates a new network which will be used as an input to the flow algorithm on the next iteration of the IIA. A discussion of why only requirements missions are considered appears in Section V.3. The general approach of the schedule improvement algorithm is to shift each mission in the schedule by an amount which leads to an improved flow pattern. The steps of the algorithm are shown in Figure 13 with a brief description of what each step does. In Step 1 the algorithm determines the amount of the time shift, which is implemented in Step 2. Step 3 determines whether the shift

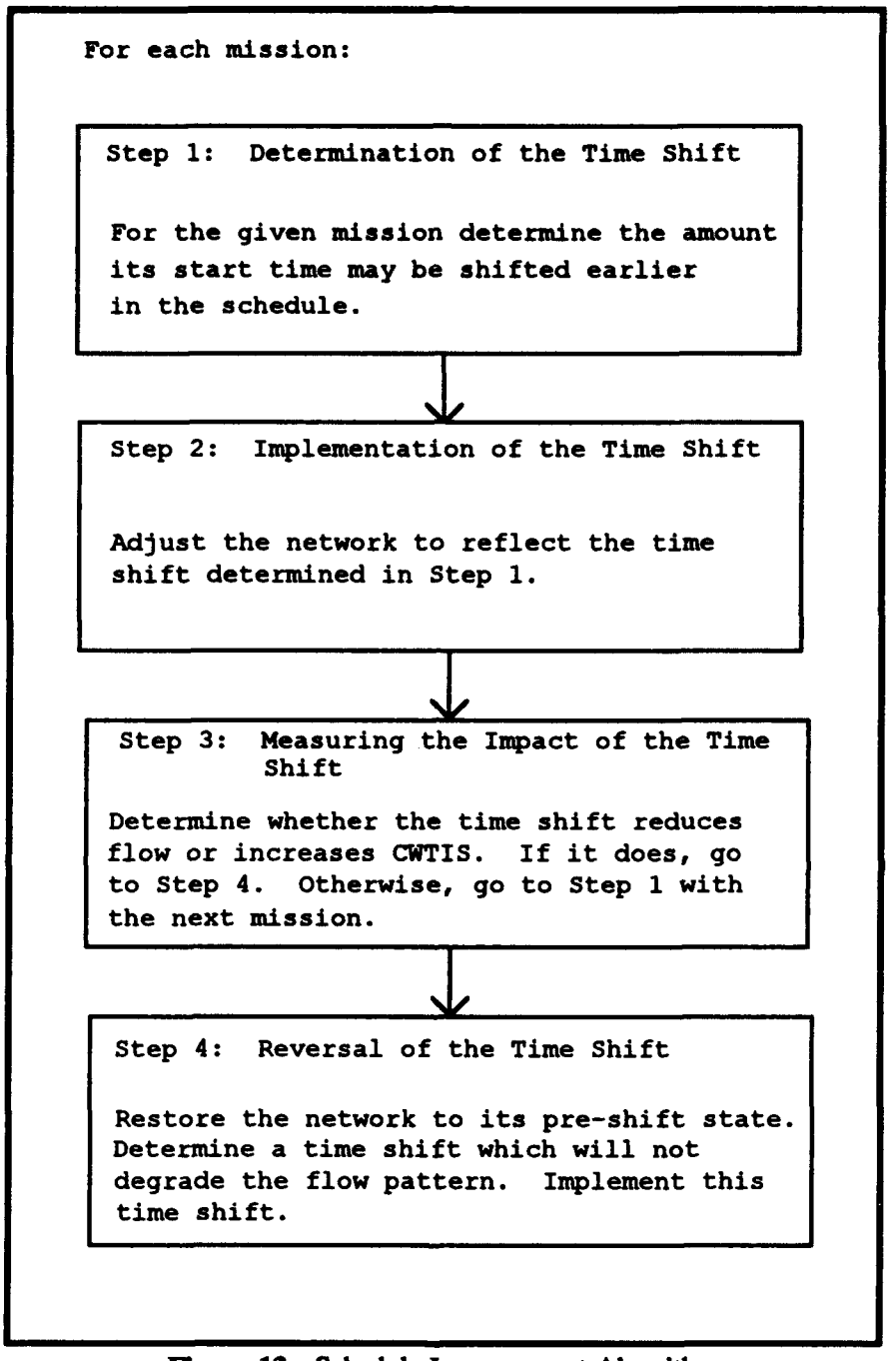


Figure 13. Schedule Improvement Algorithm

improves or degrades the flow pattern, while Step 4 reverses the time shift if the flow pattern was degraded. A detailed discussion of each step follows.

Step 1: Determination of the Time Shift. If the start time of a mission is to be changed, the magnitude of the change must be determined. Recall the two primary types of arcs within the network: mission arcs, which connect nodes on the same mission to each other, and transshipment arcs, which connect same-ICAO nodes on different missions. Arcs originating at cargo generation nodes are affected by shifts in the mission start times in the same way that transshipment arcs are affected. So, for the sake of simplicity, the definition of transshipment arcs is *extended to include both types of arcs*. Since a mission arc never becomes infeasible by implementing a time shift, attention is focused only on the transshipment arcs. Attention is further limited to *only those transshipment arcs that have positive flow*. These are derived from the set of paths found by the flow algorithm.

While a change is sought which will reduce the CWTIS of the flow pattern, the network should not be over-perturbed. A greedy approach might shift the mission by the maximum amount possible, i.e. shift its start time to time 0. While this may be successful in some instances, in most realistic cases the system would be perturbed too drastically, either causing so many transshipment arcs to become infeasible that some cargo must go undelivered or forcing a flow pattern with higher CWTIS to be used. Instead, the method developed here uses a more conservative approach, which is discussed in the following paragraphs.

If a transshipment arc terminates at a node along a mission, that arc is said to *terminate on that mission*. Likewise, a transshipment arc which originates at a node on a mission is said to *originate on that mission*. Each mission has a set of positive-flow transshipment arcs terminating on it which are sensitive to any changes in the mission start times. As the mission is shifted to an earlier time, the costs of these transshipment arcs become smaller until they become infeasible. If a time shift were chosen so that no transshipment arcs become infeasible, cargo would never have to be reflowed on different paths. Since this approach explores the possibility of reflow, time shifts are allowed which cause arc infeasibilities. But in order to

not perturb the network excessively, *only one arc is allowed to become infeasible per time shift.*

For each mission the terminating transshipment arc which is the first arc to become infeasible with a time shift is found. It is called the *most critical arc*. As multiple arcs may have the same costs, there may exist more than one most critical arc. In order for this arc to become infeasible, the time shift must be greater than its cost. Of the remaining transshipment arcs the next to become infeasible is determined. It is called the *next most critical arc*. The time shift cannot exceed the cost of this arc. If it does, more than one transshipment arc will become infeasible. If only one transshipment arc connects to the mission, the algorithm artificially defines a next most critical arc with a cost equal to the cost of the most critical arc plus a set increment.

Since the mission cannot be shifted by an amount which causes the mission start time to become a negative number, the mission start time forms an upper bound for the time shift. The time shift is defined as the minimum of {the mission start time, the cost of the next most critical arc}. Figure 14 demonstrates this process. In the figure, the vertical arcs represent three transshipment arcs terminating at a node of the mission, with costs 0.1, 0.2, and 0.3 days. The transshipment arc with cost 0.1 terminating on node B will be the first to become infeasible if the mission start time at node A is shifted sufficiently. It is the most critical arc. The transshipment arc terminating at node C with a cost of 0.2 will be the next to become infeasible. It is the next most critical arc. Since the mission start time at node A is day 1.5, the time shift equals the minimum of {1.5, 0.2}, or 0.2 days.

If the time shift is less than the cost of the most critical arc, then shifting the mission start time will not cause any transshipment arcs to become infeasible. Otherwise, the time shift represents *the maximum amount of time the mission start time can be shifted while only causing*

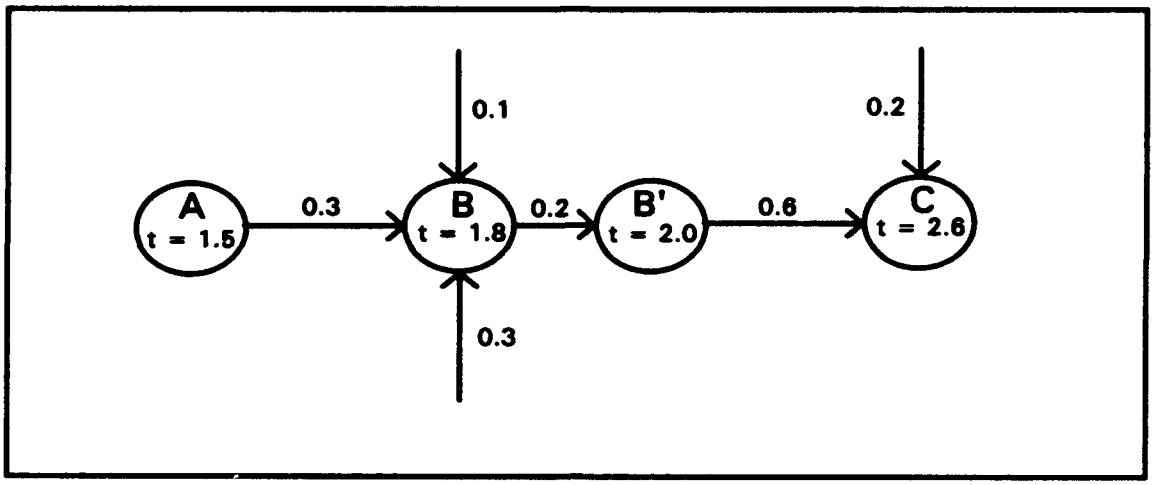


Figure 14. Determination of the Time Shift

one set of arcs, the most critical arcs, to become infeasible. After the time shift is determined, the algorithm proceeds to Step 2.

Step 2: Implementation of the Time Shift. Any change to a mission start time changes the network. The time associated with every node on that mission changes, as do the costs of every transshipment arc originating or terminating on it. As the schedule changes, the network must be updated to reflect the change. Before implementing the time shift, it is not known that it improves the flow pattern. It may become impossible to reflow some of the cargo, or the CWTIS may increase. Because of this, the shift of the mission start time will not become permanent until conditions, discussed below in Step 3, are satisfied. Instead, temporary storage arrays are maintained to store the current state of the network in case the pre-shift state must be restored later. This includes the paths used in the flow pattern, the flow and capacity of each arc in the network, and the times associated with each node.

The implementation of the mission's time shift is performed by subtracting the amount of the time shift from the times associated with each node along the mission. If the time shift is greater than zero, the state of the network has changed, requiring that the costs of the

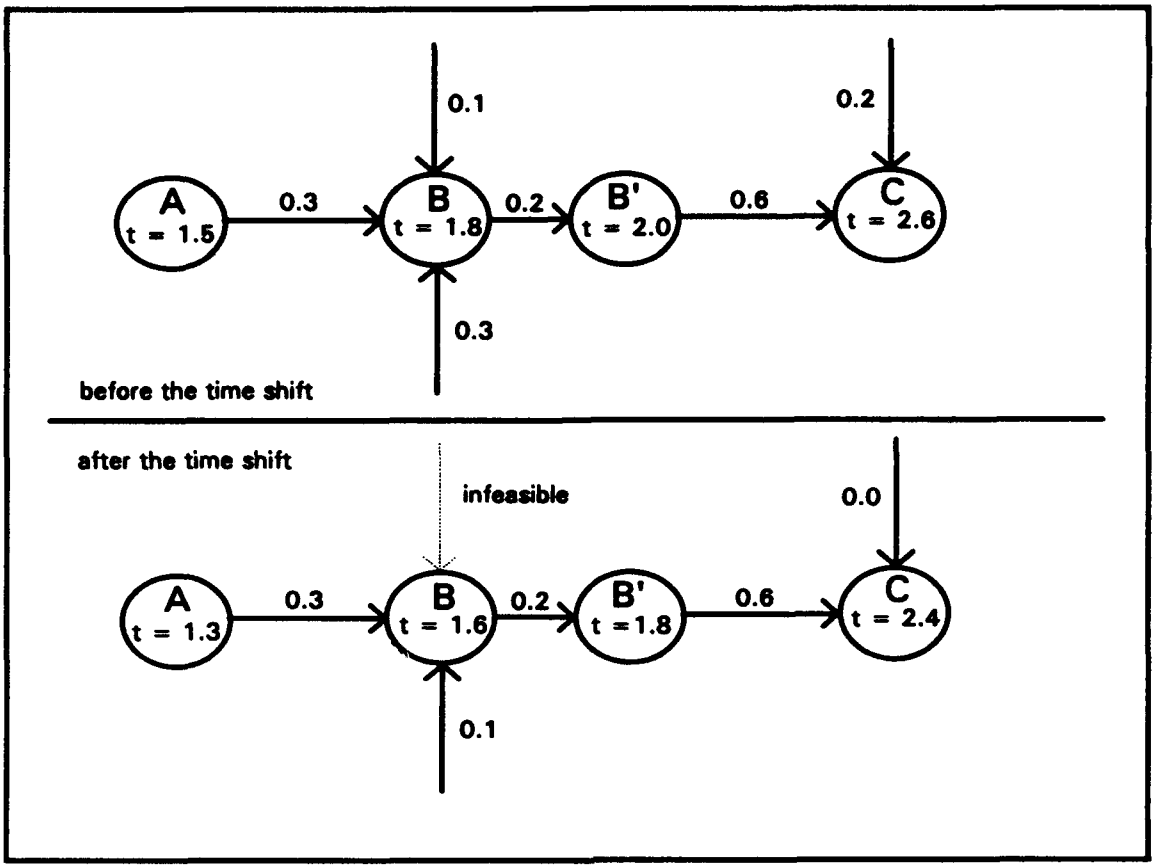


Figure 15. The Network Before and After a Time Shift of 0.2 Days

transshipment arcs be updated to reflect the time shift. However, not all of the transshipment arc costs have changed. Only those transshipment arcs which terminate on the mission or originate on the mission need to be updated. Figure 15 shows the state of a hypothetical network before and after a time shift of 0.2 days. The most critical arc, shown as a dotted line, has become infeasible and the costs associated with all other transshipment arcs have changed.

If the time shift did not cause any arc infeasibilities, the algorithm makes the changes to the network permanent and starts over with a new mission at Step 1. Otherwise, it proceeds to Step 3, discussed below.

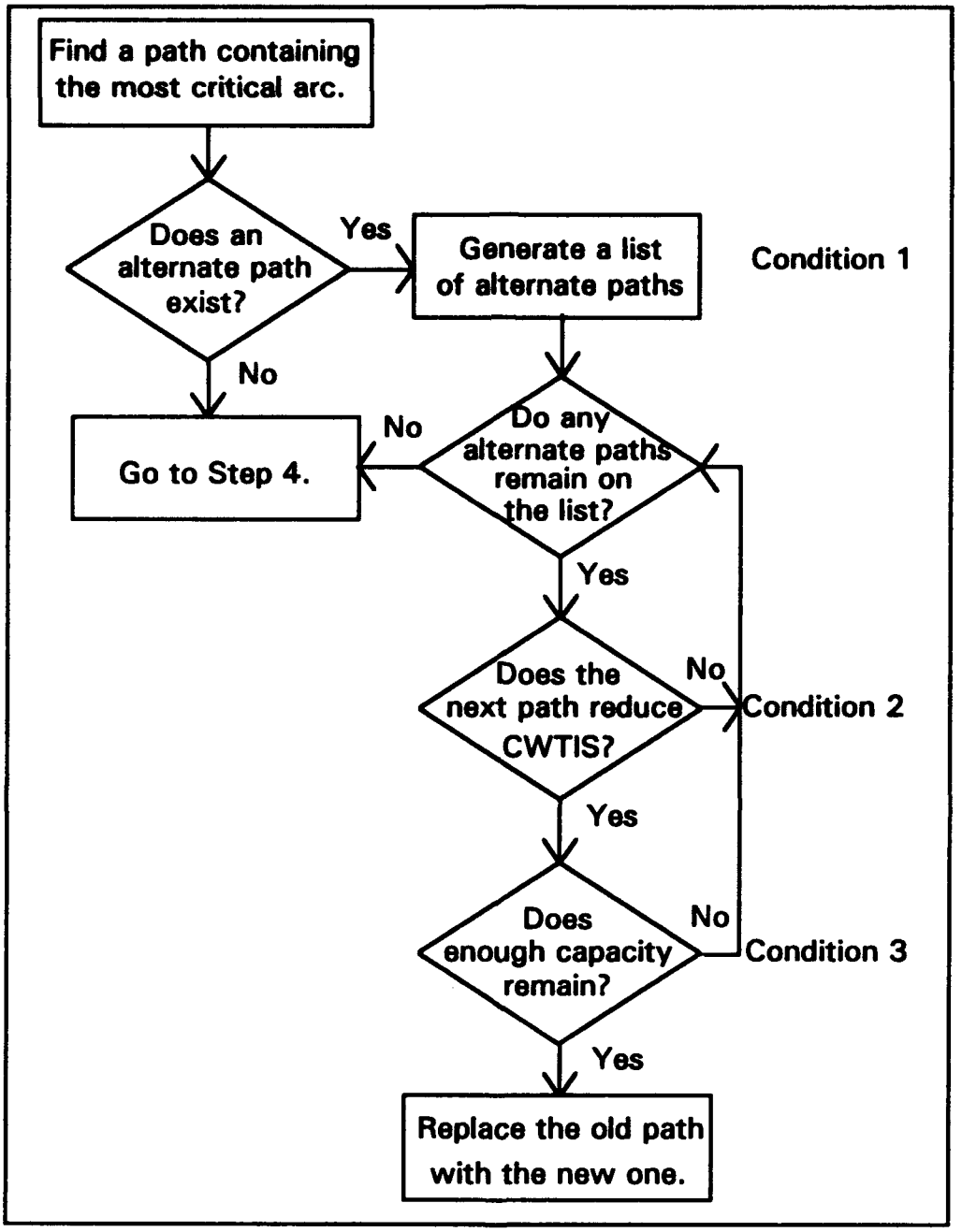


Figure 16. Reflow Conditions

Step 3: Measuring the Impact of the Time Shift. A change to the network has been implemented that could either improve or degrade the flow pattern of the channel cargo. The impact of the change must now be determined.

Since this step has been reached, the time shift has forced a transshipment arc to become infeasible. Every commodity which flowed over that arc must be reflowed on a different path which does not contain the most critical arc.

In order for a commodity to be reflowed, three conditions must be satisfied: (1) an alternate path must exist over which the cargo can be flowed, (2) using this alternate path maintains or reduces the CWTIS of the flow pattern, and (3) there must exist enough remaining capacity on the alternate path to handle the reflowed commodity. If any *one* of these three conditions is not satisfied, then the algorithm will not reflow the commodity and the time shift is deemed inappropriate. The flow chart in Figure 16 demonstrates these checks. Each condition is discussed in detail in the following paragraphs.

Checking Condition 1: The check of the impact of the shift begins by finding all the paths which contain the most critical arc. The cargo flowed on these paths must be reflowed on alternate paths since their original paths now contain an infeasible arc. Given a piece of cargo that flowed along a single path, it is possible to split the cargo into several pieces, in which case a *set* of alternate paths must be found. However, *the schedule improvement algorithm does not permit such division*. Instead, for each piece of cargo that must be reflowed, a *single* alternate path must be found to replace the original. It is left to future research to explore dividing the cargo and flowing it over several alternate paths.
If no alternate path exists to reflow any piece of cargo, implementing the time shift forces that cargo to go undelivered. Since Condition 1 has been violated, the algorithm proceeds to Step 4. If an alternate path does exist, Dijkstra's S-P algorithm returns a list of several candidate alternate paths to choose from. Considering each of them in the order Dijkstra's S-P provides, the algorithm proceeds to a check of Condition 2.

Checking Condition 2: Considering one of the candidate alternate paths produced during the check on Condition 1, it must be determined whether reflowing along the candidate path maintains or reduces CWTIS. CWTIS is likely to change. All cargo flowed

along paths that reached their sink nodes directly from the shifted mission now reach the customer sooner, so their WTISs are reduced. The cargo which flowed over the alternate path, however, may have a higher WTIS than prior to the shift. The CWTIS of the pre-shift network has been stored and is compared to the CWTIS of the post-shift network. If the post-shift CWTIS is less than or equal to the pre-shift CWTIS, it is concluded that using the alternate path does not degrade the flow pattern and the algorithm proceeds to the check on Condition 3. If the CWTIS increases, the algorithm returns to Dijkstra's list of candidate paths and continues to look for one which satisfies Condition 2. If none can be found, the algorithm leaves Step 3 and proceeds to Step 4.

Checking Condition 3: By satisfying Conditions 1 and 2, an alternate path has been found whose usage maintains or reduces CWTIS. This alternate path consists of arcs which may have been used elsewhere. In order to re-route the cargo over this path, enough capacity must exist on *each* of the arcs along it to handle the *entire* amount of the commodity being re-routed. If a single arc cannot handle the flow, the algorithm returns to Dijkstra's list of candidate paths and continues to look for one which satisfies Conditions 2 and 3. If none can be found, the algorithm leaves Step 3 and proceeds to Step 4.

If all conditions are satisfied, the original path over which the piece of cargo flowed is temporarily replaced by the alternate path. The substitution is temporary pending the check of Conditions 1, 2, and 3 for all other pieces of cargo which must be reflowed. If any piece of cargo cannot be reflowed along a different path, the time shift degrades the network flow pattern and the algorithm proceeds to Step 4. However, if all the affected cargo can be reflowed, the time shift and changes to the path set become permanent and the algorithm starts over with a new mission at Step 1.

Step 4: Reversal of the Time Shift. If a time shift degrades the flow pattern, failing one of the above three conditions, the network must be restored to the pre-shift state stored in temporary arrays. The mission start time may still be shifted by an amount that will not cause

a transshipment arc to become infeasible. Such a time shift will equal the minimum of {the mission start time, the cost of the most critical arc}. Since implementing this time shift causes no transshipment arcs to become infeasible, there is no need to check to see if any cargo needs to be reflowed. The algorithm then continues with the next mission at Step 1.

A single pass through the schedule improvement algorithm represents applying Steps 1 through 4 to the mission set *once*. After the user-specified number of passes (discussed below in Section V.4.2), the algorithm generates the new schedule to be used in the next iteration of the iterative improvement algorithm, along with distributions for the time-in-system for each piece of flowed cargo and for the number of transshipments along each path used in the flow.

V.3 Requirements vs. Frequency Missions

Recall that there are two types of AMC channel missions. These are the requirements missions and frequency-of-visit missions. As mentioned in Section V.2, the schedule improvement algorithm only considers requirements missions when manipulating the schedule. The remainder of this section discusses this restriction.

Frequency-of-visit missions, or simply frequency missions, are scheduled to occur at specific intervals within a planning horizon rather than when a cargo requirement is generated. If frequency missions were considered by the schedule improvement algorithm, their purpose may become lost in the process. For example, suppose an embassy requires four visits per month and these four missions are initially scheduled every seven days. If these missions were processed by the schedule improvement algorithm, all four missions could possibly be shifted to the first week of the month or even the first day. This would clearly be in conflict with the purpose of the frequency missions. To prevent this from happening, before the schedule improvement algorithm shifts a mission's start time, it first confirms that the mission is indeed a requirements mission. Frequency missions are left unchanged.

V.4 User-specified Parameters

The final solution is sensitive to certain parameters that are in the user's control. Specifically, the user can choose *the order in which the missions are shifted* and *the number of passes through the schedule improvement algorithm.*

V.4.1 Mission Order

The schedule improvement algorithm is a series of steps that are *sequentially* applied to the mission set. The order of the missions exhibits some influence on the final solution, so it is imperative to order them effectively. There are four orders from which the user can choose: 1) default, which is the order provided by STORM and CARGPREP, 2) reverse of the default, 3) descending mission utilization, and 4) ascending mission utilization. *Mission utilization* is a weighted measure of how much of a mission's capacity was used. Chapter VI discusses the role of this parameter with respect to the E/SWA sub-problem used in this research.

V.4.2 Passes Per Iteration of the Schedule Improvement Algorithm

Steps 1 through 4 discussed in Section V.2 represent a single pass through the mission set. At the end of this pass, the solution is guaranteed to result in an equal or reduced CWTIS for the flow pattern. However, a single pass does not make use of the information that is constantly developing as mission start times are changed. Changes that are made to missions later in the sequence may potentially create the possibility of additional changes to earlier missions. Making multiple passes through the mission set allows more information to be used, potentially leading to more improvement in the schedule. However, multiple passes obviously require more time to accomplish, although each subsequent pass may not necessarily require the same amount of time as the previous pass. The user must decide the trade-off between the time required to make multiple passes and the additional improvement achieved by them. The influence of this parameter is discussed in Chapter VI.

VI. Results

VI.1 General

This chapter presents key results which demonstrate the viability of the iterative improvement algorithm as discussed in previous chapters. The algorithm was implemented in FORTRAN and the code for this program is contained in Appendix B. Input files for this program are contained in Appendices H-K along with a discussion of their use.

VI.2 Testing Strategy

This section discusses the testing strategy used in this research, addressing the objectives of the testing as well as the design of the testing scheme.

VI.2.1 Objectives

The primary objective of the testing was to demonstrate that the iterative improvement algorithm converges on a good flow pattern for AMC's channel cargo system.

An additional objective was to provide justification for the reflow feature of the schedule improvement algorithm. Recall that one of the essential differences between this algorithm and the mathematical programming approach by Rau is the ability to reflow cargo over different paths if doing so improves the flow pattern. Testing sought to demonstrate that the reflow capability provides additional improvement. Finally, parametric analysis was conducted on the user-defined parameters, such as the cargo flow priority, the order of the missions, and the number of passes per iteration of the schedule improvement algorithm, discussed in previous chapters. Testing shows the sensitivity of the final solution to adjustments of these parameters.

VI.2.2 Design of the Testing

In order to conduct parametric analysis, several sets of runs were performed, varying the individual parameters as necessary. The primary parameters of interest were 1) the cargo flow priority (the order in which commodities were flowed), 2) the order in which the missions are examined in the schedule improvement algorithm, and 3) the number of passes per iteration of the schedule improvement algorithm. The runs within each set have identical parameter settings except for the parameter of interest.

While the reflow capability of the schedule improvement algorithm is not a user-defined parameter, several pairs of runs of the program were performed to gauge its impact. The first run in the pair allowed for cargo reflow; the second, using the same parameters as the first, was run with the reflow mechanism disabled. Later sections graphically show the results.

To analyze these parameters, over 30 runs of the computer program were made. The runs and their specific parameters are shown in Appendix 0.

The data used for testing was the same data set used in Del Rosario's research: the E/SWA sub-problem, representing 20 distinct commodities (229.99 tons total) arriving at the origin airbases over a one-week period. The initial schedule for this sub-problem, shown in Appendix I, was generated using output from AMC's STORM and CARGPREP models. It consists of 213 missions, requiring over 2,100 nodes in the network.

VI.3 Convergence of the Iterative Improvement Algorithm

The cargo flow and schedule improvement heuristics performed as expected in terms of an iterative procedure that reduced the CWTIS. The procedure continued to iterate until either the CWTIS could not be decreased or the overall quantity of cargo flow decreased. Using Unix FORTRAN on a Sun workstation, overall run times ranged from 15 minutes to approximately 2 hours.

The iterative improvement algorithm, in most cases, converged to a final solution with a CWTIS of about 50% of the initial solution and all of the cargo flowed. Figure 17 demonstrates the reduction of CWTIS during iterations for test run number 8, which yielded the lowest final CWTIS for total cargo flow (229.99 tons). On each successive iteration, the improvement in CWTIS for the flow and schedule improvement algorithms tends to decrease, indicating that as the process iterates, CWTIS becomes less sensitive to modifications of the schedule. The final solution, with significant reduction in CWTIS, demonstrates a noticeable shift in the distribution of tonnage versus TIS as seen in Figure 18. In all test runs (with reflow capability in the schedule improvement algorithm enabled), the mode of the tonnage for the distributions shifted from 1-2 days in the initial solution to 0-1 day in the final solution. Note that the lowest UMMIPS time standard for these O-D pairs is four calendar days (Robinson, 20 September 1993). Even in the initial solution, derived solely from the flow algorithm, the quantity of cargo violating UMMIPS standards was less than five percent (assuming a worst case UMMIPS standard of four days for all O-D pairs).

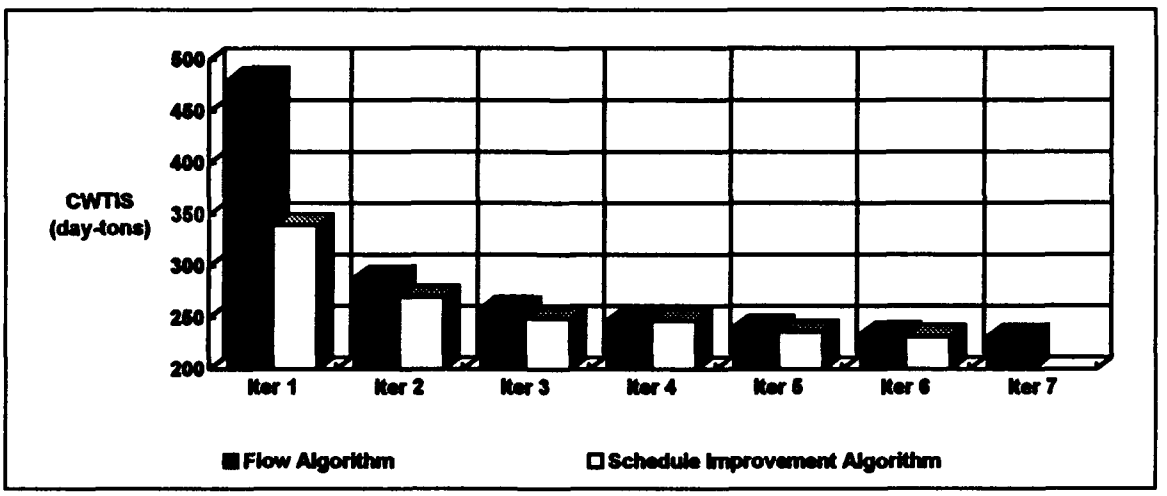


Figure 17. Reduction in CWTIS for Test Run 8

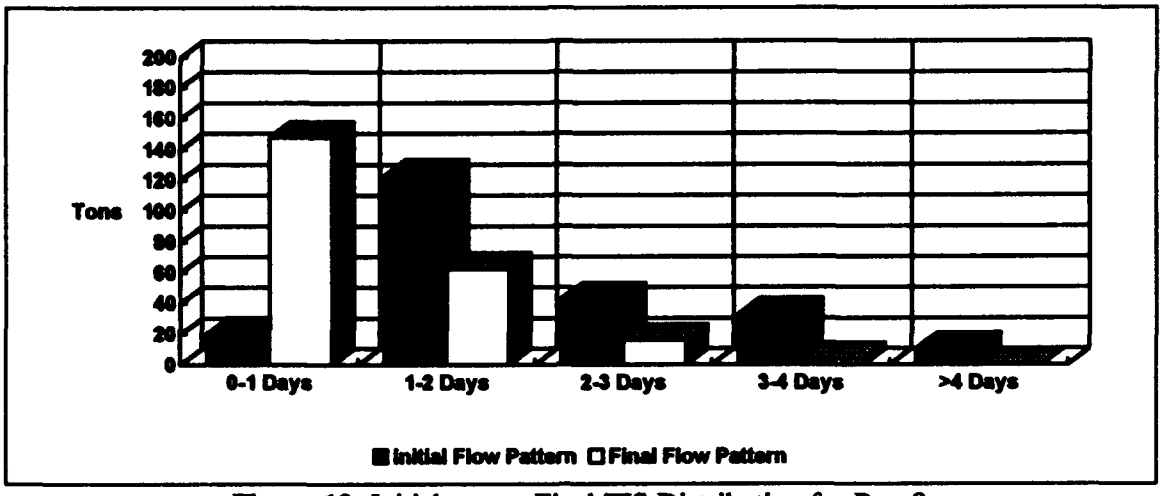


Figure 18. Initial versus Final TIS Distribution for Run 8

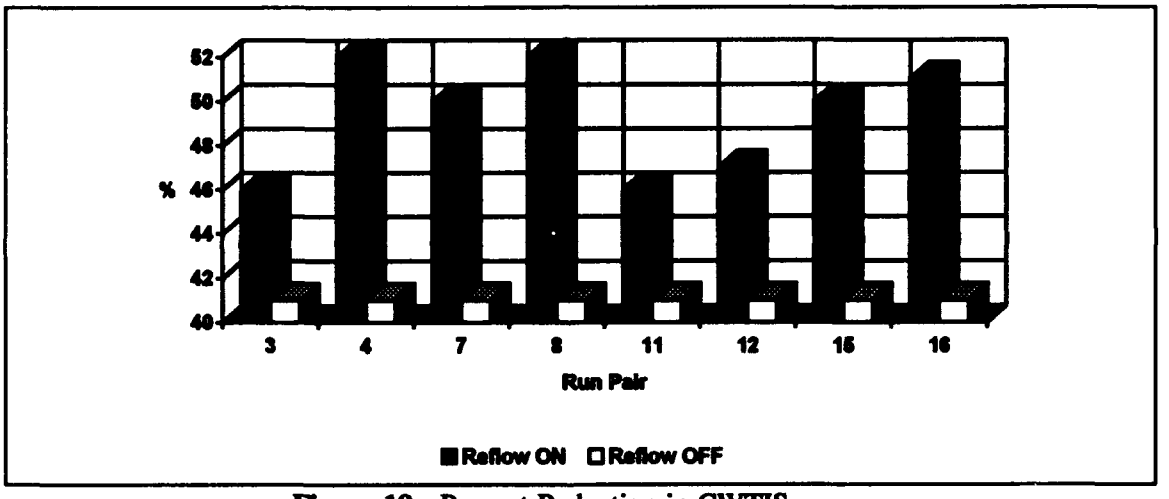


Figure 19. Percent Reduction in CWTIS

VI.4 Justification for Reflowing Cargo

Reflowing cargo in the schedule improvement algorithm can be a very involved process. This process can only be justified if it achieves a significantly improved flow pattern.

Sixteen pairs of test runs were performed (Appendix O). Each pair consists of one run with cargo reflow and one run with the reflow mechanism disabled. Figures 19 through 21 show the impact of the reflow capability. Figure 19 shows the total percent reduction in CWTIS for those runs in which the entire amount of cargo was flowed. As the figure

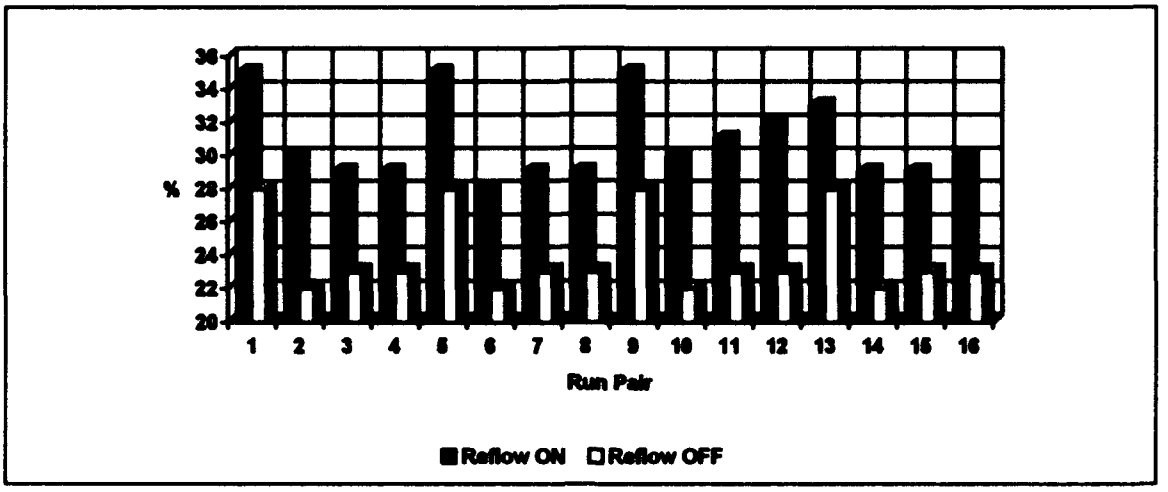


Figure 20. Percent Reduction in CWTIS on the First Iteration

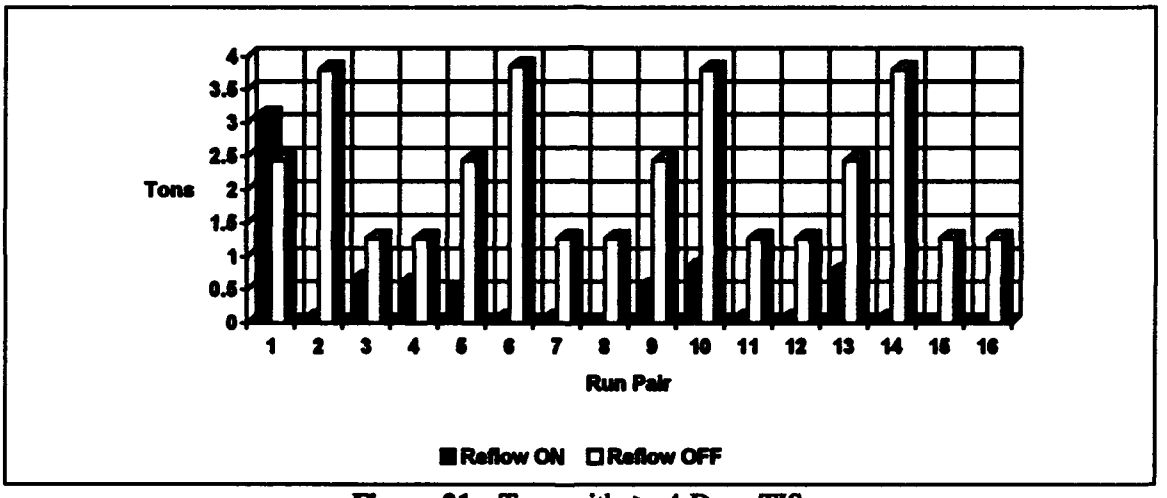


Figure 21. Tons with > 4 Days TIS

indicates, allowing cargo to be reflowed results in approximately a 5-10% additional reduction in the CWTIS.

As Figure 17 shows, for Run 8 the largest reduction in CWTIS occurred on the first iteration. This held true for all the test runs. Figure 20 shows that with the reflow mechanism disabled, the amount of reduction in CWTIS on the first iteration is reduced by about 8%.

Finally, Figure 21 shows how the lack of a reflow capability may result in more cargo possibly violating UMMIPS standards, assuming that the commodities in this sub-problem are

subject to a UMMIPS standard of four calendar days. This is a direct result of how the reflow capability influences the time-in-system distribution of the commodities. In all test cases, enabling the reflow capability shifted the mode of the time-in-system distribution from 1-2 days to 0-1 days. However, in all test cases, the mode did not shift with the reflow mechanism disabled. It is concluded that the reflow approach employed by the schedule improvement algorithm provides enough additional benefit to warrant its use.

VI.5 Flowed Cargo

The user-defined parameters affected the amount of cargo flowed by the cargo flow algorithm. The total amount of cargo in the E/SWA sub-problem was 229.99 tons. Not all of the runs were able to flow the entire amount. As shown in Figure 22, the amount of cargo flowed ranged from a low of 202.78 tons in Run 9 to a high of 229.99 tons.

Recall that the iterative improvement algorithm continues to iterate only when improvement is achieved. In all test runs in which all of the cargo (229.99 tons) was flowed, that level was reached on the first iteration and was maintained throughout the remaining iterations. In the test runs in which a lesser amount was delivered, the tonnage tended to grow from iteration to iteration, indicating that changes made in the schedule improvement algorithm changed the network sufficiently to allow the flow of additional cargo on subsequent iterations. For example, Figure 23 shows how the cargo flowed on Run 5 grew throughout that run's four iterations.

VI.6 Reflowed Cargo

The amount of cargo that was reflowed during each iteration of the flow/schedule improvement algorithm varied greatly from run to run and from iteration to iteration within each run in response to the user-specified parameters. Regardless of the parameters, the first iteration always experienced the greatest reduction in CWTIS. It is not surprising that the

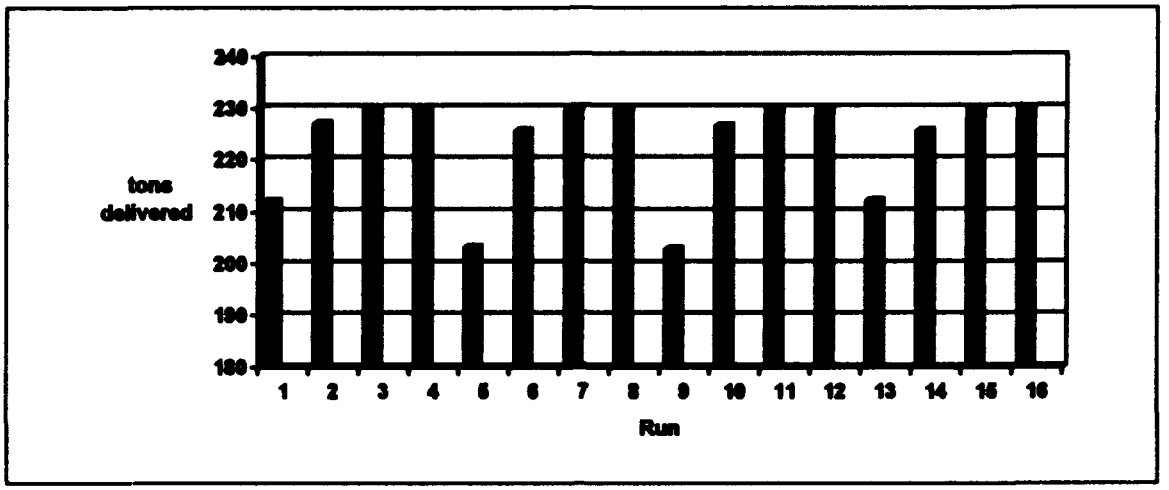


Figure 22. Cargo Flowed

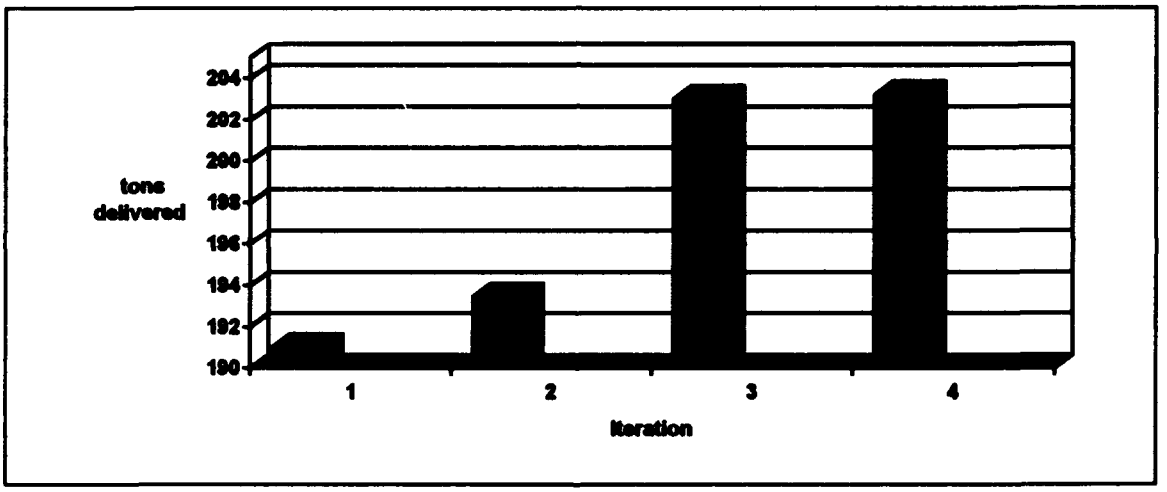


Figure 23. Cargo Flowed on Run 5

first iteration typically, though not always, experienced the largest amount of cargo reflow during Step Two. Figure 24 displays the amount of cargo reflowed on the first iteration for each run using the E/SWA sub-problem. The flow values range from a high of 70.35 tons in Run 8 to a low of 7.75 tons in Run 13. The chart values represent the percentage of the total cargo reflowed.

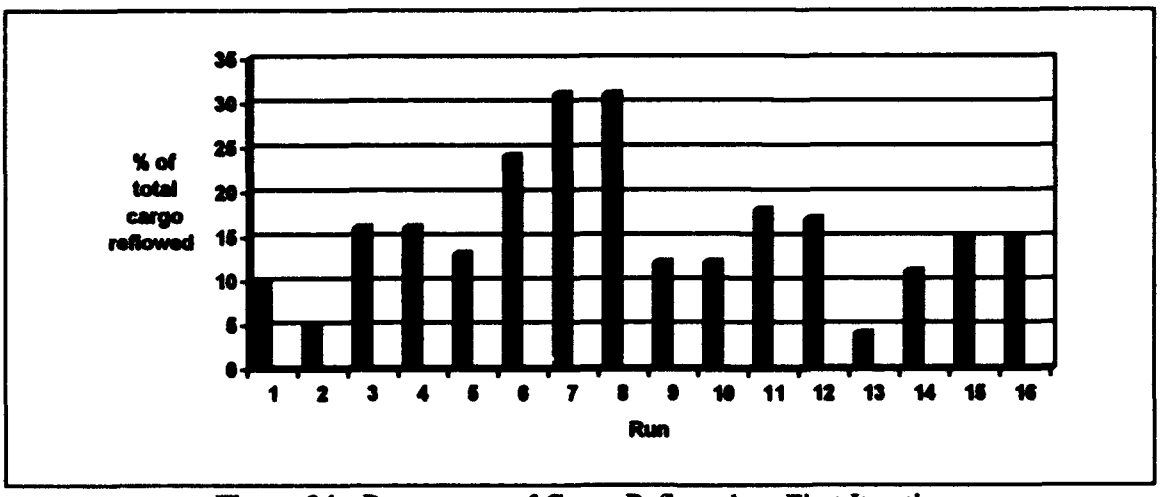


Figure 24. Percentages of Cargo Reflowed on First Iteration

VI.7 Parametric Analysis

This section addresses the influence on the final solution of the following parameters: 1) cargo flow priority, 2) mission order, and 3) the number of passes per iteration of the schedule improvement algorithm.

VI.7.1 Cargo Flow Priority

Equal cargo priority is *not possible* due to the successive nature of the flow algorithm. The cargo nodes that are selected first by the algorithm have a higher probability of being flowed with lower TIS compared to cargo nodes that are flowed later. This is due to less available capacity on the channel missions as more and more cargo is flowed. This phenomenon is called *cargo preference*, since certain cargo appears to receive preferential treatment by the cargo flow algorithm. Given O-D pairs with relatively small initial cargo quantities, the order of cargo flow in the algorithm becomes less significant since less network capacity is used. For the original E/SWA sub-problem with normal cargo levels, the order of cargo flow had no effect on the initial cumulative flow quantity (all cargo was flowed). With runs of the E/SWA sub-problem with the same flight schedule and collection of commodities, but *all cargo quantities multiplied by a factor of ten* (resulting in total cargo equal to 2299.90

tons), the effect of cargo flow order became readily apparent. These runs (Runs 33-35) are described in Appendix O. The results are summarized in Table 1.

Options for cargo flow priority include: 1) arbitrary, which is the order the commodities are listed in the cargo input file, 2) first in-first out (FIFO) based on time, and 3) progressing from the largest quantities to the smallest.

Table 1

The Effect of Cargo Flow Priority on the Final Flow Pattern

Run Number	Cargo Flow Priority	Final Flow	Final CWTIS
1	default	2115.1	18151
2	FIFO	2298.9	21974
3	quantity	2227.7	20170

When the commodities were flowed according to their location in the cargo input file, the final flow pattern did not flow all of the cargo. Many of the last commodities selected could not be flowed due to the high utilization of the channel missions. When the cargo nodes were flowed according to quantity (largest quantities first), the flow quantity was improved over the default. *When the cargo nodes were flowed according to time (first in, first flowed), the flow quantity was greatest.*

The difference among these options on cumulative flow quantity and CWTIS has been observed to increase as cargo quantities increase. The FIFO criteria, though allowing the most cargo flow in this test case, cannot be guaranteed as superior in all cases.

A possible benefit of cargo preference is the capability to flow any *high priority cargo* as expeditiously as possible. We can create a single cargo generation node with that high

priority commodity's characteristics. By flowing it through a network in which no other commodities have been flowed, the user can determine the fastest possible routing of that commodity to get it to its destination. This capability would be beneficial when flowing small size/weight commodities that are mission essential. For example, a small electronic component could be flowed independently under the assumption that its size and weight are negligible to the flow of the other commodities.

VI.7.2 Mission Order

The choice of the mission order exerted some influence on the final solution. This section explores this influence by comparing runs with various mission sorting criteria. Using Runs 1 - 16 as shown in Appendix O, we created four sets of runs: Set 1 = {Runs 1, 5, 9, and 13}, Set 2 = {Runs 2, 6, 10, and 14}, Set 3 = {Runs 3, 7, 11, and 15}, and Set 4 = {Runs 4, 8, 12, and 16}.

Although the sets differ in various parameter settings, the four runs within each set differ only in the *mission order parameter*. The first run within each set sorts the missions according to the default (the order provided by STORM and CARGPREP). The second run within each set sorts the missions in the reverse order relative to the default. The third run within each set sorts the missions according to descending mission utilization. The last run within each set sorts the missions according to ascending mission utilization.

Figures 25 through 28 show how the delivered cargo and associated CWTIS differ within Sets 1, 2, 3, and 4, respectively. Table 2 shows the best and worst mission orders within each set. In some sets, there are multiple entries in these two categories. Some runs within the sets were essentially equal, making it misleading to identify only one order as the best or worst. For example, in Set 1 the best order is listed as either the default order or the order based on ascending utilization, corresponding to Runs 1 and 13, respectively. Run 1 achieved a flow of 211.93 tons, compared to 212.07 tons in Run 13. Since these amounts are

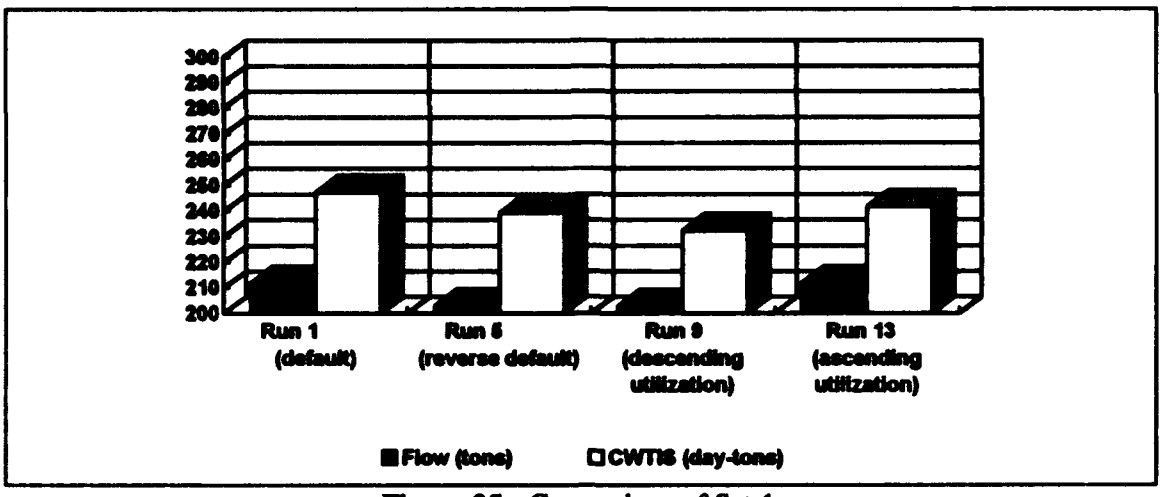


Figure 25. Comparison of Set 1

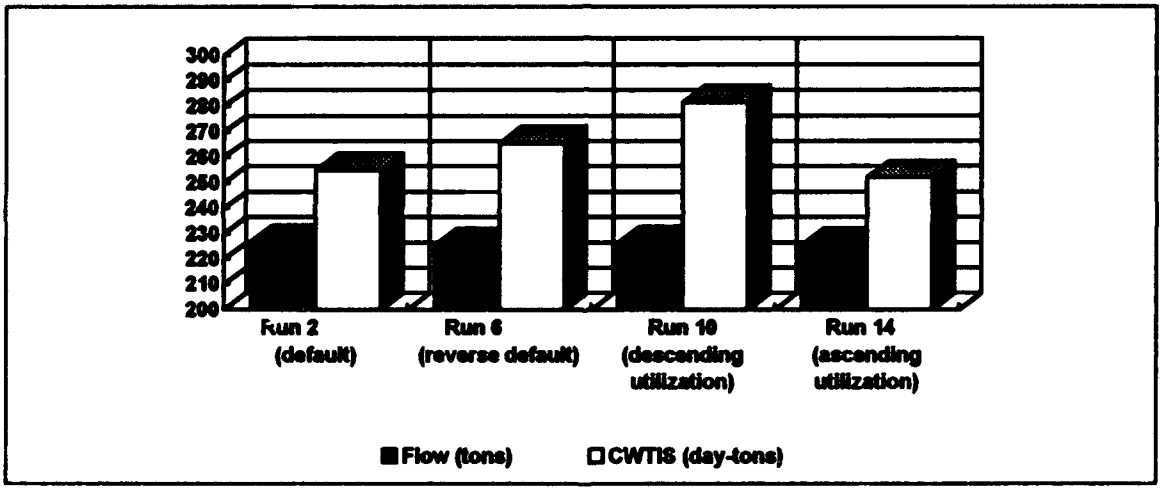


Figure 26. Comparison of Set 2

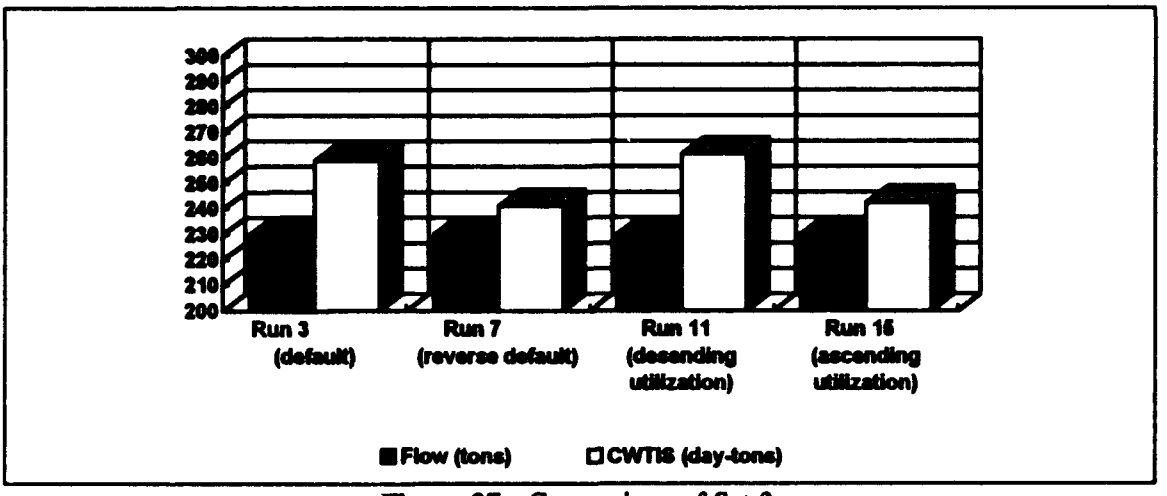


Figure 27. Comparison of Set 3

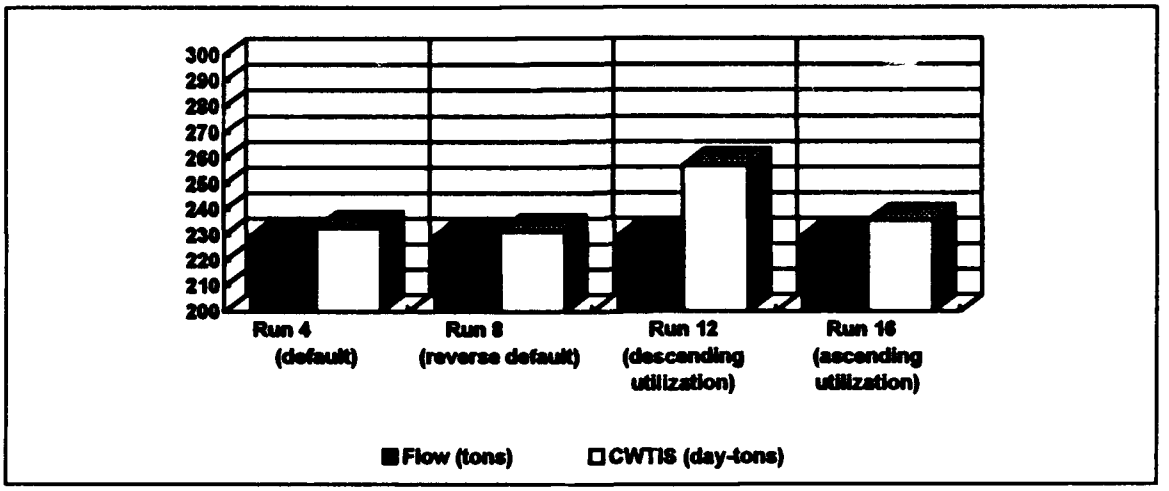


Figure 28. Comparison of Set 4

virtually equal, we looked at CWTIS to further distinguish the two runs. Run 1 achieved a CWTIS of 247.32 day-tons, while Run 13 achieved 241.72 day-tons. Again, these values are virtually equal, making it difficult to label either as the definitive best.

Table 2

Best and Worst Mission Orders

Set	Best Mission Order	Worst Mission Order
1	default or ascending utilization	reverse default or descending utilization
2	default or ascending utilization	reverse default or descending utilization
3	reverse default or ascending utilization	default or descending utilization
4	default or reverse default or ascending utilization	descending utilization

From the table, we see that ascending mission utilization achieved a better flow pattern than descending utilization in all cases. Additionally, *ascending utilization seems to be the superior method for mission ordering with these data sets*. These results are based on the E/SWA sub-problem and cannot be extended with certainty to other data sets without further testing.

VI.7.3 Passes Per Iteration of the Schedule Improvement Algorithm

Testing established that allowing the algorithm multiple passes leads to a better final solution than the final solution with only a single pass of the schedule improvement algorithm. This section presents, through a comparison of key runs, the magnitude of the solution difference when using multiple versus single passes through the schedule improvement algorithm.

As mentioned earlier, the best solution occurred on Run 8, which allowed multiple passes. Run 8 was regenerated with the pass parameter reset to 1 (a single pass). The results of the two runs are compared in Table 3 below. As the table reveals, increasing the running time of the program resulted in an additional 7% reduction in CWTIS.

Table 3

Influence of the Number of Passes on the Solution

Run	Passes	Iterations Required	Time (relative)	Final Flow	Final CWTIS	reduction in CWTIS
8	multiple	6.5	1.00	229.99	231.44	52%
8'	single	4.5	0.25	229.99	262.97	45%

We now examine in more detail the flow patterns produced by these two test runs. Table 4 and Figure 29 show the time-in-system distributions of the cargo flow for the two runs. With multiple passes, the CWTIS is reduced by an additional 7% over the single pass. This directly equates to more cargo spending less time in the system. As the table shows, 64% of the cargo spends less than one day in the system in the multiple-pass scenario, compared to only 51% in the single-pass scenario. Furthermore, the single-pass scenario has 0.83 tons of cargo spending more than four days in the system, potentially violating UMMIPS standards. By sacrificing running time, the user can obtain a substantially improved schedule with the multiple pass option. It should be noted that the user sets a *maximum* number of passes per iteration. In the run above, this was set to 10, but no more than five passes were ever required on any iteration.

Table 4

TIS Distributions for the Multiple- and Single-Pass Run 8

Multiple Pass

Days in System	Tonnage Delivered	% Total delivered
0 - 1	148.04	64%
1 - 2	62.46	27%
2 - 3	15.77	7%
3 - 4	3.72	2%
4 - 5	0.00	0%

Single Pass

Days in System	Tonnage Delivered	% Total Delivered
0 - 1	116.81	51%
1 - 2	86.44	38%
2 - 3	21.16	9%
3 - 4	4.75	2%
4 - 5	0.83	< 1%

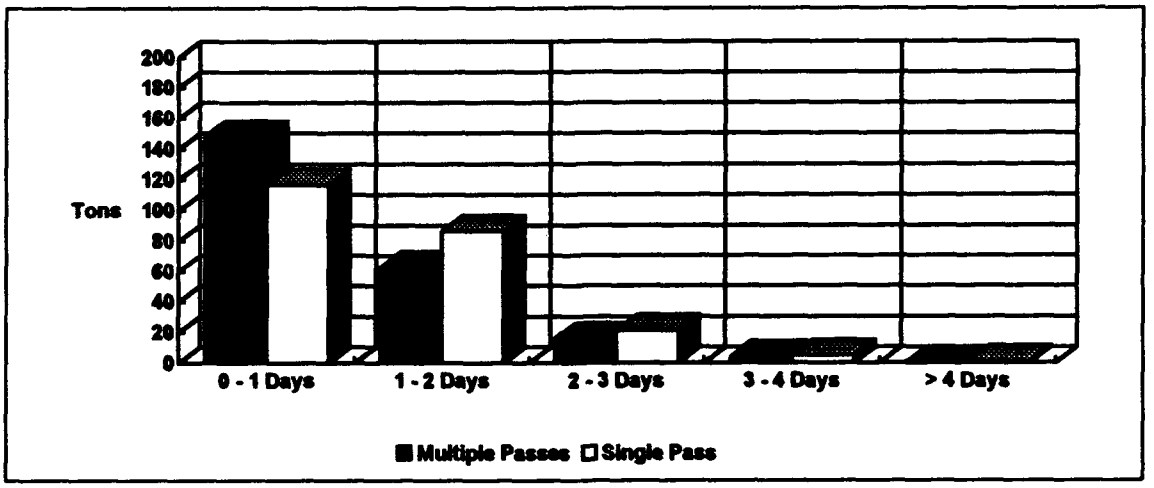


Figure 29. TIS Distributions for Run 8

VII. Conclusions and Recommendations

VII.1 General

The overall research objective, which was to develop an iterative process for efficiently scheduling airlift and flowing cargo, was achieved. The iterative improvement algorithm, consisting of the heuristic flow and schedule improvement steps, effectively increases the level of channel cargo flow while reducing CWTIS.

This chapter discusses some of the strengths of the iterative improvement algorithm as well as possible avenues for improvement or modification.

VII.2 Strengths of the Iterative Improvement Algorithm

The iterative improvement algorithm provides a *timely solution*. For the E/SWA sub-problem with 20 commodities and 213 missions, overall run times ranged from approximately fifteen minutes to two hours. This time varies with certain parameters: the maximum number of transshipments allowed, the relative quantities of the individual cargo nodes, and the number of passes through the schedule improvement algorithm per iteration. All of the test runs were accomplished on a Sun workstation using Unix FORTRAN.

The iterative improvement algorithm is *highly compatible* with the current AMC scheduling process, making validation with CARGOSIM fairly straightforward. FORTRAN code was written to preprocess STORM and CARGPREP output into data formats that are input directly into the iterative improvement algorithm. This code is listed in Appendix C.

VII.3 Recommendations for Future Research

As discussed in Chapter IV, the cargo flow heuristic follows the basic steps of the Busacker-Gowen min-cost flow algorithm, excluding the addition of reverse arcs into the network. These arcs were deemed detrimental in terms of computational efficiency and flow path control. If these reverse arcs could be introduced into the network without producing

these negative effects, the solution of the cargo flow heuristic could be improved. This addition would necessitate modifications to the S-P algorithm since Dijkstra's algorithm can only process nonnegative arc costs.

The flow algorithm terminates calls to Dijkstra's S-P algorithm whenever a path is found which exceeds the maximum allowable number of transshipments. This means that the current commodity will get no further attempts at flow paths of longer TIS with possibly fewer transshipments. The code could be extended to cover this contingency.

The flow of a commodity is "yes-or-no" and is predicated on *determination of a path from origin to destination*. Consideration should be given to flowing these goods to an alternate destination. For example, if AMC were attempting to deliver a piece of cargo from the Continental United States (CONUS) to the European theater, say Dover AFB to Rhein-Main AB, the S-P algorithm might return a "no" condition, implying that there are no paths (i.e. missions) that can get the cargo there within the current planning horizon. Certainly, getting the cargo to another, alternate destination in close proximity to the original destination would seem preferable to leaving it in the CONUS. Once in theater, other modes of distribution, such as ground transportation, could be used. In AMC's case, transporting the cargo to Ramstein, Bitburg, etc. could be the next best thing; at least the cargo is closer to its destination if it is carried over into the next planning horizon. The methodology does not currently attempt this, since no alternate destination data was available. However, the extensions to the code would be minimal.

The network, if the parameters allow for it, will generate transshipment arcs of zero time length. In real life, this would translate into a frantic loadmaster, unable to download and upload cargo in zero time. This condition was allowed to exist due to the uncertainty of many of the model parameters. Takeoff times, as well as flying times between airbases, are approximate expected values subject to variation. Additionally, aircraft commanders, if given knowledge of the upcoming requirement for cargo download, can often adjust flying times

appropriately when needed. This is considered a minor drawback of the algorithms. If this assumption is not acceptable, the network generator can be changed to establish transshipment arcs only when a minimum time differential is present.

An assumption in Chapter I stating that airbases can handle unlimited aircraft and cargo and are available 24 hours a day may not be realistic. The schedule improvement algorithm could, if conditions were right, shift the mission set so that particular airbases could become overwhelmed with aircraft. AMC refers to this as a MOG violation, where MOG is the *maximum number of aircraft on the ground that an airbase is equipped to handle*. There currently is no feasibility check within the algorithm to prevent this phenomenon. The algorithm could also reschedule a mission for any time of day, which may be unrealistic. Both of these limitations could be addressed in future research at AFIT or AMC.

The schedule improvement algorithm only implements time shifts to an *earlier* time. While it seems almost counter-intuitive to shift a mission to a later time in order to reduce the CWTIS, such a possibility exists. In the current methodology, a reduced CWTIS does not necessarily mean that *all* the cargo reached the customer sooner. While some cargo admittedly may reach a customer later because of the time shift, the savings the shift brings to the overall flow pattern may warrant such a shift.

VII.4 Validation

Following this research, the next logical step is for AMC or AFIT to test this approach in conjunction with STORM, CARGPREP, and CARGOSIM. The testing should evaluate the quality of the output schedule and the usefulness of the procedure developed in this research. Appendix A outlines all necessary steps and data structures to implement and integrate the iterative improvement algorithm within AMC's current advance planning process.

Appendix A: Program Execution Guide

This appendix contains the program execution guide, which is intended to aid the user in compiling, running, and interpreting the output of the iterative improvement algorithm (main program, **Appendix B**) developed in this research. Data input/output files and structures are addressed as necessary and appendix reference is provided, if applicable.

The main program was developed to interface with the STORM and CARGPREP output. To facilitate this, a FORTRAN program "makesked.f" (**Appendix C**) was written to preprocess the STORM and CARGPREP output and generate the "schedule.dat" file that is input into the main program. Makesked.f requires the following input files: schedule.raw (**Appendix D**), jet.dat (**Appendix E**), fly.dat (**Appendix F**), and routes.dat (**Appendix G**).

The main program requires the following input files : cargo.dat (**Appendix H**), schedule.dat (**Appendix I**), param.dat (**Appendix J**), and trnbases.dat (**Appendix K**).

Once all of the necessary files (Appendices B-K and Appendix P) have been placed into the current working directory, perform the following steps:

1) Compile and run the "makesked.f" program.

2) Edit the "param.dat" file to select the required values for the user-specified parameters.

3) Compile and run the main program ("iterate.f").

4) The user can monitor the main program through on-screen output. The current iteration and subroutine are displayed.

5) At program termination, the following output files will be present ("xxx" prior to the ".c" extension will contain the iteration number (e.g. "001") when the file was created): postxxx.c (**Appendix** [illegible]hsxxx.c (**Appendix M**), run.c (**Appendix N**), cflowxxx.c (**Appendix L**), network.dat, iterate.out, alt.out, alt2.out, paths.out, and count.out. Some of the output files are used by the program during execution and are not shown in an appendix. The purpose and format of these files are discussed within the main program comments.

6) To convert the schedule created by "iterate.f" into the format required for validation by CARGOSIM, compile and run the program "makeraw.f" (**Appendix P**).

Appendix B: Main Program Listing

This appendix contains the main program listing written in Unix FORTRAN 77. The program contains the iterative improvement algorithm as well as all supporting subroutines, including the cargo flow heuristic ("CARGFLO") and the schedule improvement heuristic ("MODMSN"). Instructions for compiling and running this program are contained in the *Program Execution Guide* at Appendix A.

```
   PROGRAM ITERATE.F
************************************************************
* ITERATIVE IMPROVEMENT ALGORITHM
*
* MAIN PROGRAM CODE
*
************************************************************

   INTEGER N,NUMCAR,NUMSNK,NUMMSN,ITER,FLONUM(4999)
   INTEGER POINT(4999,2)
   INTEGER ARCNUM
   INTEGER MAXIT,MAXALT,MAXTRN,CARCRI,PASSES
   INTEGER TRNDIS(60),TRNS(500)
   INTEGER SORCRI,TRNSHIP,TERCRI

   REAL TOTFLO,TOTAL,BEST,INF,MODFLO
   REAL COSCAP(89999,8),NODES(4999,4),TNODES(4999,4)
   REAL PRED(4999),DIST(4999),INF
   REAL EPSILON,TISDIS(60),TIMEPS

   CHARACTER NODIKO(4999,2)*4,SKEDLN*80

   COMMON /CONSTS/ N, INF, NUMCAR, NUMSNK, NUMMSN, ITER,EPSILON
   COMMON /PARAMS/ MAXIT,MAXALT,MAXTRN,CARCRI,PASSES,SORCRI,TIMEPS
   COMMON /ARRS/ COSCAP, POINT, NODES, NODIKO, TNODES
   COMMON /FLOW/ PRED, DIST, ARCNUM, TRNDIS, TRNS, TISDIS
   COMMON /SCHED/ FLONUM

C  THE FOLLOWING ALLOWS schedule.dat TO REMAIN THE SAME.  THE SUBROUTINES
C  WILL USE newsched.dat.

   OPEN(UNIT=1,FILE='schedule.dat',STATUS='OLD',FORM='FORMATTED')
   OPEN(UNIT=2,FILE='newsched.dat',STATUS='UNKNOWN',FORM='FORMATTED')
   OPEN(UNIT=7,FILE='run.c',STATUS='UNKNOWN',FORM='FORMATTED')

   DO 1 I=1,999999999
    READ(1,3,END=2)SKEDLN
    WRITE(2,3)SKEDLN
3    FORMAT(A80)
1   CONTINUE
2   CLOSE(1)
```

```
      CLOSE(2)
C  FOLLOWING WILL READ IN param.dat values
      OPEN(UNIT=1,FILE='param.dat',STATUS='OLD',FORM='FORMATTED')
       READ(1,*) MAXIT
       READ(1,*) MAXALT
       READ(1,*) MAXTRN
       READ(1,*) CARCRI
       READ(1,*) SORCRI
       READ(1,*) PASSES
       READ(1,*) TRNSHIP
       READ(1,*) EPSILON
       READ(1,*) TIMEPS
       WRITE(7,*)'Parameters:'
       WRITE(7,*)
       WRITE(7,11)MAXIT
11     FORMAT(I4,4X,'MAXIT')
       WRITE(7,12)MAXALT
12     FORMAT(I4,4X,'MAXALT')
       WRITE(7,13)MAXTRN
13     FORMAT(I4,4X,'MAXTRN')
       WRITE(7,14)CARCRI
14     FORMAT(I4,4X,'CARCRI')
       WRITE(7,16)SORCRI
16     FORMAT(I4,4X,'SORCRI')
       WRITE(7,17)PASSES
17     FORMAT(I4,4X,'PASSES')
       WRITE(7,18)TRNSHIP
18     FORMAT(I4,4X,'TRNSHIP')
       WRITE(7,19)EPSILON
19     FORMAT(F5.3,3X,'EPSILON')
       WRITE(7,21)TIMEPS
21     FORMAT(F5.4,3X,'TIME EPSILON')
       WRITE(7,*)
       WRITE(7,*)
      CLOSE(1)
      INF=99999999.9
      TOTFLO=0.0
      TOTAL=0.0
      BEST=INF
      MODFLO=0.0
      ITER=1
      OPEN(UNIT=10,FILE='iterate.out',STATUS='UNKNOWN',FORM='FORMATTED')
      WRITE(10,*)
      WRITE(10,*)'>>>>>> iterate.out <<<<<<'
      WRITE(10,*)
      WRITE(*,*)'NETMAKE CALLED'
5     CALL NETMAKE(TRNSHIP)
      WRITE(*,*)'CARGFLO CALLED'
      WRITE(7,*)'CARGFLO CALLED'
      CALL CARGFLO(TOTFLO,TOTAL)
      WRITE(*,*)TOTFLO,TOTAL
      WRITE(7,*)TOTFLO,TOTAL
```

```
C  CRITERION FOR TERMINATION
   IF((ABS(TOTFLO-MODFLO).LT.EPSILON)) THEN
     IF((TOTAL.GT.BEST)) GOTO 10
     IF(ABS(TOTAL-BEST).LT.EPSILON) GOTO 10
   ENDIF
   IF(TOTFLO.LT.MODFLO) GOTO 10
   IF (ITER.GT.MAXIT) THEN
    WRITE(10,*)
    WRITE(10,*)' MAX NUMBER OF ITERATIONS EXCEEDED.'
    GOTO 10
   ENDIF
   WRITE(*,*)'POSTPROC CALLED'
   CALL POSTPROC()
   WRITE(*,*)'COUNTER CALLED'
   CALL COUNTER()
   WRITE(*,*)'PREMOD CALLED'
   CALL PREMOD()
   WRITE(*,*)'MODMSN CALLED'
   WRITE(7,*)'MODMSN CALLED'
   CALL MODMSN(BEST,MODFLO,TOTAL,TERCRI)
   WRITE(*,*)MODFLO,BEST
   WRITE(7,*)MODFLO,BEST
   IF(TERCRI.EQ.1) GOTO 10
   WRITE(10,*)
   WRITE(10,*)' ITERATION: ',ITER
   WRITE(10,*)'  CARGO FLOWED (TONS): ',TOTFLO
   WRITE(10,*)'  TOTAL CHANNEL COST (DAY-TONS): ',TOTAL
   WRITE(*,*)'ITERATION ',ITER,' COMPLETED.'
   WRITE(*,*)' '
   WRITE(7,*)'ITERATION ',ITER,' COMPLETED.'
   WRITE(7,*)' '
   ITER=ITER+1
   GOTO 5
10  WRITE(10,*)
   WRITE(10,*)' NO FURTHER IMPROVEMENT - TERMINATED.'
   CLOSE(10)
20  STOP
   END

c NETWORK GENERATOR ALGORITHM  FOR THE CHANNEL CARGO SYSTEM
c
c PROGRAM WILL READ DATA FILES 'cargo.dat' AND
c 'newsched.dat' AND GENERATE THE NETWORK TO
c INPUT INTO THE BUSACKER-GOWEN MINIMUM COST FLOW
c ALGORITHM.  OUTPUT FROM PROGRAM WILL GO INTO FILE
c 'network.dat', WHICH WILL HAVE A STANDARDIZED FORMAT:
c
c VARIABLES USED:
c
c N  : NUMBER OF NODES IN THE NETWORK
c NUMCAR  : NUMBER OF CARGO NODES IN THE NETWORK
c NUMSINK  : NUMBER OF SINK NODES IN THE NETWORK
```

```
c INF: VALUE OF INFINITY USED IN ARRAYS
C
c COSCAP: Nx8 MATRIX OF ARC COSTS/CAPACITIES/FLOWS
c       AS WELL AS TEMP STORAGE FOR THESE VALUES
c  COSCAP IS A MODIFIED 'LINKED ADJACENCY LIST' THAT
c  LISTS THE FORWARD STAR FOR ALL NODES WITHIN THE NETWORK
C   ARRAY 'POINT' WILL BE A POINTER ARRAY FOR THE STAR FROM
C   THE SPECIFIED NODE (SEE BELOW). COSCAP IS FORMATTED:
C
C  COSCAP(X,1): END NODE OF ARC X
C  COSCAP(X,2): COST (TIME) OF GIVEN ARC
C  COSCAP(X,3): CAPACITY (TONS) OF GIVEN ARC
C  COSCAP(X,4): FLOW (TONS) OF GIVEN ARC
C  COSCAP(X,5): TEMPORARY COST (TIME) OF GIVEN ARC
C  COSCAP(X,6): TEMPORARY CAPACITY (TONS) OF GIVEN ARC
C  COSCAP(X,7): TEMPORARY FLOW (TONS) OF GIVEN ARC
C  COSCAP(X,8): MISSION (#) WHICH ARC TERMINATES TO
C
C  POINT: Nx2 MATRIX OF POINTER LOCATIONS FOR NODES:
C
C  POINT(Y,1): FIRST ARC LOCATION IN COSCAP FOR NODE Y
C  POINT(Y,2): LAST ARC LOCATION IN COSCAP FOR NODE Y
C
C  IF A GIVEN NODE HAS NO FORWARD STAR (NO ARCS BEGINNING
C  THERE), THEN POINT(N,1),POINT(N,2) WILL EQUAL -1.
C
c NODES : Nx4 MATRIX DESCRIBING NODE SET IN NETWORK
c      (N,1) COLUMN IS NODE DESCRIPTOR, WHERE:
c       -1 : CARGO GENERATION NODE AT N
c        0 : SINK NODE AT NODE N
c        1 : NODE N REPRESENTS A MISSION AIRBASE ORIGIN
C          (FIRST BASE FOR THE ROUTE)
c        2 : NODE N REPRESENTS AN INTERMEDIATE MISSION AIRBASE
c        3 : NODE N REPRESENTS A MISSION AIRBASE DESTINATION
C          (LAST BASE FOR THE ROUTE)
c      (N,2) COLUMN IS NODE TIME
c      (N,3) COLUMN IS TONS OF CARGO FOR A CARGO GEN. NODE
c          ACCAPA STORED HERE FOR MISSION LEGS BETWEEN
C          DISTINCT AIRBASES, INF FOR GROUND/RON TIME
C          DURING A MISSION
c      (N,4) COLUMN IS THE MISSION NUMBER ASSOCIATED WITH
C          THE GIVEN AIRBASE (0 FOR SOURCE/SINK NODES)
C          MISSION NUMBERS ASSIGNED FROM 001 ASCENDING
C
c NODIKO : Nx2 MATRIX (BOTH FIELDS CHARACTER) DESCRIBING NODES
c      (N,1) COLUMN IS ORIGIN AIRBASE IDENTIFIER FOR A
c          CARGO GENERATION NODE (N,1)=-1 ABOVE
c          THIS WILL ALSO APPLY FOR A SINK NODE (N,1)=0
c          WHERE THE VALUE IN (N,3) WILL BE THE NUMBER
c          IDENTIFIER OF THE AIRBASE THE NODE SINKS FOR.
c          FOR A MISSION AIRBASE, THIS WILL BE ITS ICAO.
c      (N,2) COLUMN IS DESTINATION AIRBASE IDENTIFIER FOR
```

```
c          A CARGO GENERATION NODE ABOVE
c
c ACTYPE : AIRCRAFT TYPE IDENTIFIER (CHARACTER*4) (I.E. C141)
c ACCAPA : CORRESPONDING AIRCRAFT CAPACITY IN TONS (I.E. 20)
c       (THIS WILL BE MISSION LEG ARC CAPACITY)
c ARCNUM : COUNTER FOR NUMBER OF ARCS CREATED THUS FAR

      SUBROUTINE NETMAKE(TRNSHIP)

      INTEGER N,K,TEMP
      INTEGER NUMCAR,NUMSNK,NUMMSN,N,ITER,ARCNUM
      INTEGER WEEK,UNIQUE
      INTEGER POINT(4999,2)
      INTEGER TRNDIS(60),TRNS(500)
      INTEGER TRNSHIP,NOTRBS,NUMNOD

      REAL COSCAP(89999,8)
      REAL NODES(4999,4),TNODES(4999,4)
      REAL INF,DIST(4999),PRED(4999)
      REAL ACCAPA,COSTMP,CAPTMP
      REAL TMP,DELTA
      REAL DAYCUM(8),RATIO
      REAL EPSILON,TISDIS(60)

      CHARACTER NETNAM*8,NS*3,EXT*2,POS(3)*1
      CHARACTER*4 NODIKO(4999,2)
      CHARACTER*4 ORIGIN,DEST,ACTYPE
      CHARACTER*4 TRNBAS(499)

      COMMON /CONSTS/ N, INF, NUMCAR, NUMSNK, NUMMSN, ITER,EPSILON
      COMMON /ARRS/ COSCAP, POINT, NODES, NODIKO, TNODES
      COMMON /FLOW/ PRED, DIST, ARCNUM, TRNDIS, TRNS, TISDIS

      ACCAPA=0.0
      N=0
      NUMCAR=0
      NUMSNK=0
      ARCNUM=0
      DAYCUM(1)=0.0
c  INITIALIZE DATA ARRAYS
      DO 250 I=1,89999
       DO 260 J=1,7
        COSCAP(I,J)=0.0
260    CONTINUE
250   CONTINUE
      DO 255 I=1,4999
       NODES(I,1)=0.0
       NODES(I,2)=0.0
       NODES(I,3)=0.0
       NODIKO(I,1)='    '
       NODIKO(I,2)='    '
255   CONTINUE
```

```
C  IF TRNSHIP=1, 'trnbases.dat' READ TO LIMIT TRANSSHIPMENT BASES
      IF (TRNSHIP.EQ.1) THEN
       OPEN(UNIT=1,FILE='trnbases.dat',STATUS='OLD',FORM='FORMATTED')
       READ(1,*)NOTRBS
       DO 257 I=1,NOTRBS
        READ(1,*)TRNBAS(I)
257    CONTINUE
       CLOSE(1)
      ENDIF
      OPEN(UNIT=1,FILE='cargo.dat',STATUS='OLD',FORM='FORMATTED')
C  CARGO.DAT MODIFIED FROM PREVIOUS FORMAT IN ORDER TO FACTOR THE
C  WEEK INTO THE CARGO GENERATION SCHEME.  ADDITIONAL COL. ADDED AFTER
C  THE INITIAL TWO ICAOS IN ORDER TO INDICATE WHICH WEEK OF THE PLANNING
C  HORIZON THE SEVEN SUBSEQUENT COLUMNS REPRESENTED (STARTS @ WEEK 0)
      DO 200 I=1,99999999
       READ(1,201,END=202)ORIGIN,DEST,WEEK,(DAYCUM(J),J=2,8)
201    FORMAT(A4,1X,A4,1X,I2,7(F7.2))
       DO 203 J=2,8
        IF (DAYCUM(J).GT.DAYCUM(J-1)) THEN
         N=N+1
         NODES(N,1)=-1
         NODES(N,2)=(WEEK*7.0)+(J-2)
         NODES(N,3)=DAYCUM(J)-DAYCUM(J-1)
         NODIKO(N,1)=ORIGIN
         NODIKO(N,2)=DEST
        ENDIF
203    CONTINUE
200    CONTINUE
202    CONTINUE
c  STORE WHERE REMAINDER OF NODES SHOULD PICK-UP AFTER CARGO GEN NODES
      NUMCAR=N
      OPEN(UNIT=2,FILE='newsched.dat',STATUS='OLD',FORM='FORMATTED')
      READ(2,*)NUMMSN
      DO 206 I=1,NUMMSN
       READ(2,204)NUMNOD,ACTYPE,ACCAPA
204    FORMAT(I3,1X,A4,F5.1)
C  FOLLOWING DISTINQUISHES BETWEEN MISSION AIRBASES
       DO 207 J=1,NUMNOD
        N=N+1
        IF (J.EQ.1) THEN
         NODES(N,1)=1
        ENDIF
        IF ((J.GT.1).AND.(J.LT.NUMNOD)) THEN
         NODES(N,1)=2
        ENDIF
        IF (J.EQ.NUMNOD) THEN
         NODES(N,1)=3
        ENDIF
        READ(2,*)NODIKO(N,1),NODES(N,2)
        NODES(N,3)=ACCAPA
        NODES(N,4)=I
207    CONTINUE
```

```
206  CONTINUE
c NOW TO PRODUCE THE SINK NODES, WHICH WILL BE DETERMINED BY SCANNING ALL
c OF THE CARGO GENERATION NODES' DESTINATIONS (NODES(N,1)=-1=>CARGO NODE),
c USE ICAO OF NODIKO(N,2) AS SINK IDENTIFIER.
c THE SINK MATRIX IS SCANNED, THEN A NEW SINK NODE IS CREATED EVERY TIME
c A DISTINCT SINK ICAO APPEARS.
    DO 209 I=1,NUMCAR
     UNIQUE=1
     DO 210 J=1,I-1
      IF (NODIKO(I,2).EQ.NODIKO(J,2)) THEN
       UNIQUE=0
      ENDIF
210   CONTINUE
c CREATE DISTINCT SINK NODE IF UNIQUE=1
    IF (UNIQUE.EQ.1) THEN
     NUMSNK=NUMSNK+1
     N=N+1
     NODES(N,1)=0
     NODIKO(N,1)=NODIKO(I,2)
    ENDIF
209  CONTINUE
C NOW THAT NODES/NODIKO ARE CREATED, THE LINKED ADJACENCY LIST
C NEEDS TO BE CREATED BY GOING NODE BY NODE THROUGH THE NETWORK
C (EXCLUDING THE SINK NODES AT THE BOTTOM WHICH WILL HAVE NO NODES
C ORIGINATING FROM THEM) AND LISTING INFO FOR ALL ARCS EMANATING
C FROM THE SPECIFIED NODE

    DO 350 K=1,N-NUMSNK
     POINT(K,1)=ARCNUM+1
C ADD ARCS TO CONNECT CARGO TO ORIGIN AIRBASES BELOW:
C (RETAIN ALL ARCS VERSUS SINGLE ARC FROM CARGO TO ORIGIN
C AIRBASE OF LEAST TIME, IN CASE TRANSSHIPMENTS ARE RESTRICTED
C IN THE FUTURE (I.E. CERTAIN BASES NOT ALLOWED TO TRANSSHIP))
   IF (K.GT.NUMCAR) GOTO 215
     DO 220 J=NUMCAR+1,N-NUMSNK
      IF(NODIKO(J,1).EQ.NODIKO(K,1)) THEN
       IF(NODES(J,2).GE.NODES(K,2)) THEN
        ARCNUM=ARCNUM+1
        COSCAP(ARCNUM,1)=J
        COSCAP(ARCNUM,2)=NODES(J,2)-NODES(K,2)
        COSCAP(ARCNUM,3)=INF
       ENDIF
      ENDIF
220  CONTINUE
C THIS LOOP ADDS MISSION ARCS
215  IF ((K.GT.NUMCAR).AND.(NODES(K,1).LT.3)) THEN
     ARCNUM=ARCNUM+1
     COSCAP(ARCNUM,1)=K+1
     COSCAP(ARCNUM,2)=NODES(K+1,2)-NODES(K,2)
     IF (NODIKO(K,1).NE.NODIKO(K+1,1)) THEN
      COSCAP(ARCNUM,3)=NODES(K,3)
     ELSE
```

```
      COSCAP(ARCNUM,3)=INF
     ENDIF
    ENDIF
C ADD ARCS TO ALLOW TRANSSHIPMENTS BELOW
C RESTRICTED TO A SPANNING PATH VERSUS ALL POSSIBLE
C ARCS PREVIOUSLY, WHICH WAS EXPLOSIVE.
C LOOP WILL SCAN MISSION NODES FOR A GIVEN ICAO AND
C LINK THEM IN A PATH BASED ON TIME SEQUENCE,
C NOTE: CYCLES MAY BE INTRODUCED TO THE PATH IF TWO
C AIRBASES WITH THE SAME ICAO HAVE THE SAME TIME
    IF (K.LE.NUMCAR) GOTO 245
C IF TRNSHIP=0, NO TRANSSHIPMENT ARCS ALLOWED; BYPASS.
    IF (TRNSHIP.EQ.0) GOTO 245
C IF TRNSHIP=1, SCAN TRNBAS() TO CHECK IF ALLOWED.
    IF (TRNSHIP.EQ.1) THEN
     DO 239 I=1,NOTRBS
      IF (TRNBAS(I).EQ.NODIKO(K,1)) GOTO 242
239   CONTINUE
     GOTO 245
    ENDIF
C INSTALL TRANSSIPMENT ARC
242  TMP=INF
    J=0
    DO 240 I=NUMCAR+1,N-NUMSNK
     IF (I.EQ.K) GOTO 240
     IF ((I.EQ.K+1).AND.(NODES(K,1).LT.3)) GOTO 240
     IF (NODIKO(I,1).EQ.NODIKO(K,1)) THEN
      DELTA=NODES(I,2)-NODES(K,2)
      IF (DELTA.GT.0.0) THEN
       IF (DELTA.LE.TMP) THEN
        TMP=DELTA
        J=I
       ENDIF
       GOTO 240
      ENDIF
C BASES WITH THE SAME TIMES ARE LINKED AUTOMATICALLY
C (WILL BE IN BOTH DIRECTIONS WHEN THE I-LOOP IS DONE)
      IF (DELTA.EQ.0.0) THEN
       ARCNUM=ARCNUM+1
       COSCAP(ARCNUM,1)=I
       COSCAP(ARCNUM,2)=DELTA
       COSCAP(ARCNUM,3)=INF
      ENDIF
     ENDIF
240  CONTINUE
C IF-THEN ESTABLISHES SINGLE ARC TO CLOSEST TIME AIRBASE
     IF (J.NE.0) THEN
      ARCNUM=ARCNUM+1
      COSCAP(ARCNUM,1)=J
      COSCAP(ARCNUM,2)=NODES(J,2)-NODES(K,2)
      COSCAP(ARCNUM,3)=INF
     ENDIF
```

```
c ADD 0 COST ARCS THAT CONNECT NODES TO SINKS BELOW:
245  IF (K.LE.NUMCAR) GOTO 230
     DO 235 J=N-NUMSNK+1,N
      IF (NODIKO(J,1).EQ.NODIKO(K,1)) THEN
       ARCNUM=ARCNUM+1
       COSCAP(ARCNUM,1)=J
       COSCAP(ARCNUM,2)=0.0
       COSCAP(ARCNUM,3)=INF
       GOTO 237
      ENDIF
235   CONTINUE
237   CONTINUE
230  POINT(K,2)=ARCNUM
C  IF-THEN DETERMINES IF ANY ARCS ORIGINATE FROM NODE K
    IF (POINT(K,2).LT.POINT(K,1)) THEN
     POINT(K,1)=-1
     POINT(K,2)=-1
     GOTO 350
    ENDIF
C FOLLOWING LOOP SORTS ARCS FROM GIVEN NODE (ASCENDING)
C ALL CARGO NODES WILL BE SORTED, SKIP THIS LOOP
    IF (K.LE.NUMCAR) GOTO 350
    IF (POINT(K,2).EQ.POINT(K,1)) GOTO 350
    DO 352 I=POINT(K,1),POINT(K,2)
     TEMP=-1
     TMP=INF
     DO 354 J=I,POINT(K,2)
      IF (COSCAP(J,1).LE.TMP) THEN
       TMP=COSCAP(J,1)
       TEMP=J
      ENDIF
354   CONTINUE
     COSTMP=COSCAP(TEMP,2)
     CAPTMP=COSCAP(TEMP,3)
     COSCAP(TEMP,1)=COSCAP(I,1)
     COSCAP(TEMP,2)=COSCAP(I,2)
     COSCAP(TEMP,3)=COSCAP(I,3)
     COSCAP(I,1)=TMP
     COSCAP(I,2)=COSTMP
     COSCAP(I,3)=CAPTMP
352  CONTINUE
350  CONTINUE
C  FOLLOWING LOOP STORES -1 IN POINT FOR SINKS
    DO 360 K=N-NUMSNK+1,N
     POINT(K,1)=-1
     POINT(K,2)=-1
360  CONTINUE
    OPEN(UNIT=3,FILE='network.dat',STATUS='UNKNOWN',FORM='FORMATTED')
    WRITE(3,*)N
    WRITE(3,*)NUMCAR
    WRITE(3,*)NUMSNK
    WRITE(3,*)INF
```

```
      DO 290 I=1,ARCNUM
       WRITE(3,365)I,(COSCAP(I,J),J=1,4)
365    FORMAT(I5,2x,F5.0,2x,F7.3,2x,F13.2,2x,F9.3)
290   CONTINUE
      DO 295 I=1,N
       WRITE(3,370)I,(POINT(I,J),J=1,2)
370    FORMAT(I5,3x,I5,3x,I5)
295   CONTINUE
      WRITE(3,*)
      DO 325 I=1,N
       WRITE(3,330)I,(NODES(I,J),J=1,3),(NODIKO(I,J),J=1,2)
330    FORMAT(I5,1x,F5.1,2x,F6.2,2x,F7.2,2x,A4,2x,A4)
325   CONTINUE
340   CLOSE(1)
      CLOSE(2)
      CLOSE(3)
      RETURN
      END

c     CARGO FLOW ALGORITHM
c
c  variable CARCRI used to vary cargo flow sequence:
c   CARCRI=1 ->Default, flowed in order of cargo.dat
c   CARCRI=2 ->Flowed in ascending arrival time order
c   CARCRI=3 ->Flowed in descending cargo quantity
c
c  code applies successive shortest path implementation
c  adjusting the network capacity as it goes along
c  (uses DIJKSTRA and not 'PDM' to find path)
C
c  Array TISDIS added to track distribution of cargo flow
c  based on time-in-system.

      SUBROUTINE CARGFLO(TOTFLO,TOTAL)

      INTEGER N,S,T,I,J,Q,R
      INTEGER FLPATH(4999),FLNUM,CURNODE
      INTEGER STPATH,NUMCAR,NUMSNK,NUMMSN,ITER,TEMP
      INTEGER POINT(4999,2)
      INTEGER ARCNUM,ARCTMP
      INTEGER BST,WST,LOTRN
      INTEGER CARGPT(4999),SORT(4999)
      INTEGER TRNDIS(60),TRNS(500)
      INTEGER MAXIT,MAXALT,MAXTRN,CARCRI,PASSES,SORCRI

      REAL CURREN,DELTA,D,TARGET,TOTAL,INF,RATIO
      REAL COSCAP(89999,8)
      REAL PRED(4999),DIST(4999),NODES(4999,4),TNODES(4999,4)
      REAL COMCOST,TOTFLO
      REAL CAPAIJ,FLOWIJ
      REAL EPSILON,TISDIS(60),TIMEPS
```

```
      CHARACTER*4 APOE,APOD
      CHARACTER*4 NODIKO(4999,2)
      CHARACTER MCNAM*10,NS*3,EXT*2,POS(3)*1

      COMMON /CONSTS/ N, INF, NUMCAR, NUMSNK, NUMMSN, ITER,EPSILON
      COMMON /PARAMS/ MAXIT,MAXALT,MAXTRN,CARCRI,PASSES,SORCRI,TIMEPS
      COMMON /ARRS/ COSCAP, POINT, NODES, NODIKO, TNODES
      COMMON /FLOW/ PRED, DIST, ARCNUM, TRNDIS, TRNS, TISDIS

      TOTFLO=0.0
      UNFLOW=0.0
      TOTAL=0.0
      OPEN(UNIT=4,FILE='paths.out',STATUS='UNKNOWN',FORM='FORMATTED')
      OPEN(UNIT=11,FILE='alt.out',STATUS='UNKNOWN',FORM='FORMATTED')
c FLOW/TCOST must be stored thru all commodities
      DO 450 I=1,ARCNUM
         COSCAP(I,4)=0.0
         COSCAP(I,5)=COSCAP(I,2)
         COSCAP(I,6)=COSCAP(I,3)
         COSCAP(I,7)=0.0
450   CONTINUE
      DO 452 I=1,60
       TRNDIS(I)=0
       TISDIS(I)=0.0
452   CONTINUE
c following scans network for all cargo nodes and determines
c Source (S) and Sink (T) to be used for B-G algorithm
c
c assumed that all cargo generation nodes will be the first
c nodes 1-NUMCAR in the network (this is the way 'netmake.f'
c generates them from cargo.dat and newsched.dat)
*     THE FOLLOWING LOOP DESIGNATES A UNIQUE FILENAME TO EACH ITERATION
      TEMP=ITER
      EXT='.c'
      DO 451 I=2,0,-1
       RATIO=INT(TEMP/(10**I))
       TEMP=TEMP-RATIO*(10**I)
       POS(I+1)=CHAR(48+RATIO)
451   CONTINUE
      NS=POS(3)//POS(2)//POS(1)
      MCNAM='cflow'//NS//EXT
      OPEN(UNIT=5,FILE=MCNAM,STATUS='UNKNOWN',FORM='FORMATTED')
      WRITE(5,*)'<<<<<<<<<<<<<<<  CARGFLO.OUT >>>>>>>>>>>>>>>'
      WRITE(5,*)'     '
      GO TO (420,422,424,426), CARCRI
420     WRITE(5,*)'Flow sequence based on order in cargo.dat'
      GOTO 428
422     WRITE(5,*)'Flow sequence based on arrival time'
      GOTO 428
424     WRITE(5,*)'Flow sequence based on commodity quantity'
      GOTO 428
426     WRITE(5,*)'Flow sequence based on O-D distance'
```

```
      GOTO 428
428   WRITE(5,*)'    '
      WRITE(5,*)' Total number of cargo nodes:',NUMCAR
      WRITE(5,*)'    '
C sorting routine follows to sort cargo nodes based on either
c arrival time, quantity, or o-d distance
c actual nodes won't be sorted within the network,
c CARGPT array will be used to sequence
C selection criteria is user-specified by CARCRI
      DO 429 I=1,NUMCAR
        SORT(I)=0
        CARGPT(I)=I
429   CONTINUE
      IF (CARCRI.EQ.0) GOTO 440
      DO 430 I=1,NUMCAR
       BST=INF
       WST=0.0
       DO 432 J=1,NUMCAR
       GO TO (440,434,436,438), CARCRI
434    IF ((NODES(J,2).LE.BST).AND.(SORT(J).EQ.0)) THEN
       K=J
       BST=NODES(J,2)
      ENDIF
      GOTO 432
436    IF ((NODES(J,3).GE.WST).AND.(SORT(J).EQ.0)) THEN
       K=J
       WST=NODES(J,3)
      ENDIF
      GOTO 432
C  O-D distance sorter not currently coded
438    GOTO 432
432      CONTINUE
      SORT(K)=1
      CARGPT(I)=K
430   CONTINUE
c  end of sorting, begin flowing commodities
440    DO 409 Q=1,NUMCAR
       S=CARGPT(Q)
       APOE=NODIKO(S,1)
       APOD=NODIKO(S,2)
       TARGET=NODES(S,3)
c  following loop scans network to find correct sink
      T=-1
      DO 410 R=N-NUMSNK+1,N
       IF (NODIKO(R,1).EQ.APOD) THEN
         T=R
       ENDIF
410    CONTINUE
      WRITE(5,*)'    '
      WRITE(5,*)'    '
      WRITE(5,*)' Origin-Destination ICAOs: ',APOE,'-',APOD
      WRITE(5,*)' Arrival time:',NODES(S,2),' Quantity:',TARGET
```

```
    WRITE(5,*)' Cargo node (source):',S,' Sink node:',T
C ***** CARGFLO MAIN PROGRAM FOLLOWS
    COMCOST=0.0
    CURREN=0
    STPATH=0
***** MAIN INITIALIZATION OVER *****
470   CALL DIJKSTRA(S,T,STPATH,LOTRN)
      WRITE(5,*)' '
      IF (STPATH.GT.0) THEN
c compress path prior to processing
      CALL COMPRESS(S,T,STPATH,LOTRN)
C COMPRESS MAY CHANGE THE NUMBER OF TRANSSHIPMENTS
      IF (LOTRN.GT.MAXTRN) GOTO 479
C STORE LOTRN INTO TRNDIS ARRAY
      TRNDIS(LOTRN+1)=TRNDIS(LOTRN+1)+1
       DELTA=INF
       I=T
475     IF(I.EQ.S) GOTO 480
       J=I
       I=PRED(J)
C LOOP NEEDED HERE TO FIND CAPA(I,J)
      CAPAIJ=-1.0
      FLOWIJ=-1.0
      DO 477 K=POINT(I,1),POINT(I,2)
       IF (INT(COSCAP(K,1)).EQ.J) THEN
        CAPAIJ=COSCAP(K,6)
        FLOWIJ=COSCAP(K,7)
        GOTO 478
       ENDIF
477    CONTINUE
      WRITE(*,*)'>>>> ERROR, CAPAIJ NOT FOUND.'
478     IF (CAPAIJ.GT.0.0) THEN
        D=CAPAIJ-FLOWIJ
        IF (D.LT.DELTA) DELTA=D
       ENDIF
       GOTO 475
480     IF ((CURREN+DELTA).GT.TARGET) THEN
        DELTA=TARGET-CURREN
       ENDIF
       I=T
485     IF(I.EQ.S) GOTO 490
       J=I
       I=PRED(J)
       CAPAIJ=-1.0
       FLOWIJ=-1.0
      DO 486 K=POINT(I,1),POINT(I,2)
      IF (INT(COSCAP(K,1)).EQ.J) THEN
       CAPAIJ=COSCAP(K,6)
       FLOWIJ=COSCAP(K,7)
       ARCTMP=K
       GOTO 488
      ENDIF
```

```
486    CONTINUE
       WRITE(*,*)'>>>> ERROR, CAPAIJ NOT FOUND.'
488     IF (CAPAIJ.GT.0.0) THEN
        FLOWIJ=FLOWIJ+DELTA
        COSCAP(ARCTMP,7)=FLOWIJ
C  FLOATING POINT ARITHMETIC TOLERANCE BUILT IN TO CATCH
C  ANY SMALL DEVIATIONS HERE
        IF(ABS(CAPAIJ-FLOWIJ).LE.EPSILON) COSCAP(ARCTMP,5)=INF
       ENDIF
       GOTO 485
490    CURREN=CURREN+DELTA
****** THIS SECTION WRITES OUT THE PATH A COMMODITY TAKES *****
       CURNODE=T
       FLPATH(1)=T
       FLNUM=1
        DO 495 I=2,N
         FLPATH(I)=PRED(CURNODE)
         IF(PRED(CURNODE).EQ.-1.0) GOTO 496
         CURNODE=PRED(CURNODE)
         FLNUM=FLNUM+1
495       CONTINUE
496      IF(FLNUM.NE.1) THEN
       WRITE(5,*)' TIS: ',DIST(T),' FLOW: ',DELTA
       WRITE(5,*)' NO. OF TRANSSHIPMENTS: ',LOTRN
       WRITE(5,*)' ICAO   MSN NO.  TIME   NODE NUMBER'
       WRITE(5,*)' ====   =======  =====  ==========='
       DO 500, I=FLNUM,1,-1
        J=FLPATH(I)
        WRITE(5,502)NODIKO(J,1),NODES(J,4),NODES(J,2),J
502      FORMAT(2X,A4,3X,F5.0,5X,F5.2,5X,I6)
500     CONTINUE
       WRITE(5,*)
       WRITE(4,*)DIST(T),LOTRN,DELTA,FLNUM,(FLPATH(I),I=FLNUM,1,-1)
       COMCOST=COMCOST+DIST(T)*DELTA
      ENDIF
C  STORE FLOW INTO TISDIS ARRAY
     TISDIS(INT(DIST(T))+1)=TISDIS(INT(DIST(T))+1)+DELTA
      ENDIF
***** STPATH *****
    IF ((CURREN.LT.TARGET).AND.(STPATH.GT.0)) GOTO 470
C  FOLLOWING COMPUTES TOTAL SYSTEM COST (I.E. DAY-TONS!)
C  COMCOST IS THE COST FOR THE GIVEN COMMODITY FLOW
479   TOTAL=TOTAL+COMCOST
     TOTFLO=TOTFLO+CURREN
     UNFLOW=UNFLOW+(TARGET-CURREN)
 ***** END COMMODITY FLOW *****
C ADD OUTPUT FOR TOTAL CHANNEL SYSTEM CARGO UNFLOWED
       WRITE(5,*)
       WRITE(5,*)' FLOWED CARGO FOR THIS COMMODITY:',CURREN
       WRITE(5,*)' UNFLOWED CARGO FOR THIS COMMODITY:',TARGET-CURREN
       WRITE(5,*)' COST FOR FLOW OF THIS COMMODITY:',COMCOST
       WRITE(5,*)
```

```
      WRITE(5,*)'TOTAL CHANNEL SYSTEM CARGO FLOWED:',TOTFLO
      WRITE(5,*)'TOTAL CHANNEL SYSTEM CARGO NOT FLOWED:',UNFLOW
      WRITE(5,*)'TOTAL CHANNEL SYSTEM COST:',TOTAL
409     CONTINUE
c 409 continue loop for next commodity
C  WRITE OUT TRANSSHIPMENT DISTRIBUTION
    WRITE(5,*)
    WRITE(5,*)
    WRITE(5,*)
    WRITE(5,*)' TRANSSHIPMENT DISTRIBUTION'
    WRITE(5,*)
    WRITE(5,*)' NUMBER   OCCURENCES'
    WRITE(5,*)' ----------'
    WRITE(7,*)
    WRITE(7,*)
    WRITE(7,*)
    WRITE(7,*)' TRANSSHIPMENT DISTRIBUTION'
    WRITE(7,*)
    WRITE(7,*)' NUMBER   OCCURENCES'
    WRITE(7,*)' ----------'
    DO 503 I=1,60
     IF (TRNDIS(I).GT.0) THEN
      WRITE(5,*)(I-1),'       ',TRNDIS(I)
      WRITE(7,*)(I-1),'       ',TRNDIS(I)
     ENDIF
503   CONTINUE
C  WRITE OUT T.I.S DISTRIBUTION
    WRITE(5,*)
    WRITE(5,*)
    WRITE(5,*)
    WRITE(5,*)' T.I.S. DISTRIBUTION'
    WRITE(5,*)
    WRITE(5,*)' DAYS       TONS '
    WRITE(5,*)' ----------'
    WRITE(7,*)
    WRITE(7,*)
    WRITE(7,*)
    WRITE(7,*)' T.I.S. DISTRIBUTION'
    WRITE(7,*)
    WRITE(7,*)' DAYS       TONS '
    WRITE(7,*)' ----------'
    DO 504 I=1,60
     IF (TISDIS(I).GT.0) THEN
      WRITE(5,*)(I-1),'-',I,'     ',TISDIS(I)
      WRITE(7,*)(I-1),'-',I,'     ',TISDIS(I)
     ENDIF
504   CONTINUE
    CLOSE(4)
    CLOSE(5)
    CLOSE(11)
    RETURN
    END
```

```
c DIJKSTRA'S SHORTEST PATH ALGORITHM FOR
c FLOW APPLICATIONS IN THE CHANNEL CARGO SYSTEM
c
c MODIFIED TO FIND ALTERNATE PATHS OF THE SAME LENGTH
C AND TO CHOOSE THE 'BEST' USING CRITERIA OF TRANSSHIPMENT,
C OR LEAST TIME IN-AIR, OR, ETC.
c
c Method derived from "Discrete Optimization Algorithms with Pascal
c programs" (Syslo, Deo, Kowalik)
c
c N is the number of nodes.
c S is the source node
c T is the sink node
c INF is "infinity"
c PATH is 1-0 variable that determines if there is a path from the source to
c  the sink "1" = true and "0" = false
c DIST is the array containing the shortest distance from source to nodes
c  that have been permanently labeled.
c PRED is the array a shortest path from source (node S) can be traced.
c FINAL is the array for each node determining if it has been labeled permanent
c  where "1" = permanent and "0" = temporary.
c
C TIS IS TIME-IN-SYSTEM FOR A PATH
C TIA IS TIME-IN-AIR FOR A PATH
C LOWTIA IS LOWEST TIME-IN-AIR FOR PATHS
C DELTA IS MINIMUM ARC CAPACITY ALONG A PATH
C HIDELT IS HIGHEST OVERALL PATH CAPACITY
C MAXTRN IS MAXIMUM NUMBER OF TRANSSHIPMENTS ALLOWED
C NUMTRN IS CURRENT PATH NUMBER OF TRANSSHIPMENTS
C MAXALT IS MAXIMUM NUMBER OF ALT PATHS ALLOWED
C NUMALT IS CURRENT NUMBER OF ALT PATHS FOUND
C
C CURRENT LOGIC WILL:
C 1) NUMALT.LT.MAXALT, FIND AN ALTERNATE PATH.
C 2) DETERMINE ITS NUMBER OF TRANSSHIPMENTS (NUMTRN),
C    DELTA, AND TIA AS WELL.
C 3) IF NUMTRN.GT.MAXTRN GOTO 1.
C 4) IF DELTA.LT.HIDELT GOTO 1
C 5) IF DELTA.EQ.HIDELT, RESORT TO TIME-IN-AIR AS
C    CHOOSE CRITERIA.  LOWEST TIA SELECTED,GOTO 1
C 6) IF DELTA.GT.HIDELT, STORE NEW PATH,HIDELT, GOTO 1

   SUBROUTINE DIJKSTRA(S,T,PATH,LOTRN)

   INTEGER N,S,T,PATH,FINAL(4999),RECENT,Y
   INTEGER NUMCAR,NUMSNK,NUMMSN,ITER
   INTEGER POINT(4999,2)
   INTEGER ARCNUM,NUMLBL,ENDNOD,NUMBAK
   INTEGER FLPATH(4999),FLNUM,CURNODE
   INTEGER I,J,L,M,I2
   INTEGER BSTPTH(4999),BSTNUM
```

```
      INTEGER NUMTRN,LOTRN
      INTEGER NUMALT,O,FIRST,PRIOR
      INTEGER MAXIT,MAXALT,MAXTRN,CARCRI,PASSES,SORCRI
      INTEGER TRNDIS(60),TRNS(500)

      REAL DIST(4999),PRED(4999),INF,LABEL
      REAL COSCAP(89999,8),NODES(4999,4),TNODES(4999,4)
      REAL TIS,TIA
      REAL DELTA,HIDELT,BSTDLT
      REAL COST
      REAL EPSILON,TISDIS(60),TIMEPS

      CHARACTER*4 NODIKO(4999,2)

      COMMON /CONSTS/ N, INF, NUMCAR, NUMSNK, NUMMSN, ITER,EPSILON
      COMMON /ARRS/ COSCAP, POINT, NODES, NODIKO, TNODES
      COMMON /FLOW/ PRED, DIST, ARCNUM, TRNDIS, TRNS, TISDIS
      COMMON /PARAMS/ MAXIT,MAXALT,MAXTRN,CARCRI,PASSES,SORCRI,TIMEPS

C  MAXALT AND MAXTRN AFFECT ALT PATHS AND ARE USER-SPECIFIED
      LOTRN=99
      NUMALT=0
      NUMLBL=0
      HIDELT=0.0
      BSTDLT=0.0
      DO 515 I=1,N
       DIST(I)=INF
       PRED(I)=-1.0
       FINAL(I)=0
515   CONTINUE
      DIST(S)=0.0
      FINAL(S)=1
      PATH=0
      RECENT=S
c  Dijkstra looks at forward star nodes
c  from recent instead of all nodes when
c  using the linked adjacency list
588   DO 520 I=POINT(RECENT,1),POINT(RECENT,2)
      IF (I.LT.0) GOTO 525
      IF (COSCAP(I,5).LT.INF) THEN
       IF (FINAL(INT(COSCAP(I,1))).EQ.0) THEN
        LABEL=DIST(RECENT)+COSCAP(I,5)
        IF(LABEL.LT.DIST(INT(COSCAP(I,1)))) THEN
         DIST(INT(COSCAP(I,1)))=LABEL
         PRED(INT(COSCAP(I,1)))=RECENT
        ENDIF
       ENDIF
      ENDIF
520   CONTINUE
525   TEMP=INF
C CHANGED UPDATE SCAN TO EXCLUDE CARGO NODES, THEY CANNOT
C BE LABELED BECAUSE NO ARCS END AT THEM
```

```
      DO 530 U=NUMCAR+1,N
       IF (FINAL(U).EQ.0) THEN
        IF (DIST(U).LT.TEMP) THEN
         Y=U
         TEMP=DIST(U)
        ENDIF
       ENDIF
530   CONTINUE
      IF(TEMP.LT.INF) THEN
       FINAL(Y)=1
       RECENT=Y
       NUMLBL=NUMLBL+1
        ELSE
         PATH=0
         FINAL(T)=1
         GOTO 599
        ENDIF
     IF(FINAL(T).LT.1) THEN
      GOTO 588
     ELSE
      PATH=1
     ENDIF
C ALTERNATE PATHS LOGIC FOLLOWS (NORMAL DIJKSTRA ENDS HERE)
C TAKE THE SELECTED PATH AND TRACE BACK IN
C THE PREDECESSOR ARRAY, ALL OF THESE NODES WILL BE USED AS
C END VERTICES FOR A SCAN OF ALL PERMANENTLY LABELLED NODES,
C IF AN ARC EXISTS BETWEEN A PERMANENTLY LABELLED NODE AND
C THE PREDECESSOR NODE, AN ALTERNATE PATH IS FOUND
C FILE 'alt.out' WILL CONTAIN INFO ON THE SEARCH IN FORMAT
C VERY SIMILAR TO 'paths.out':
c TIS/TIME-IN-AIR/FLOW CAPACITY/# TRANSSHIPMENTS/# NODES IN PATH/PATH
C VARIABLE 'MAXALT' USER-SET TO LIMIT NUMBER OF ALTERNATE PATHS FOUND
C VARIABLE 'MAXTRN' USER-SET TO LIMIT NUMBER OF TRANSSHIPMENTS
C ALGORITHM WILL FIND/EVALUATE PATHS UNTIL MAXALT REACHED, THEN
C THE MOST ADVANTAGEOUS PATH WILL BE SELECTED BASED ON USER
C SELECTED CRITERIA.
      NUMBAK=0
      FIRST=1
      I=T
555     IF ((I.EQ.S).OR.(NUMALT.GE.MAXALT)) GOTO 599
      J=I
      I=PRED(J)
      IF (FIRST.EQ.1) GOTO 570
575     NUMBAK=NUMBAK+1
   DO 540 K=NUMCAR+1,N-NUMSNK
* ONLY CHECK PERMANENTLY LABELED NODES
   IF (FINAL(K).LT.1) GOTO 540
* ANY NODE THAT FOLLOWS K IN THE INITIAL PATH
* (CLOSER TO THE SINK) CANNOT BE USED, OR A
* CYCLE WOULD BE INTRODUCED INTO THE PRED ARRAY.
* (THIS WILL ONLY HAPPEN WITH 0 COST ARCS)
   I2=T
```

```
      DO 560 M=1,NUMBAK+1
       I2=PRED(I2)
       IF (K.EQ.I2) GOTO 540
560   CONTINUE
*  CHECK TO SEE IF ANY ARCS ORIGINATE AT NODE K
      IF (POINT(K,1).LT.0) GOTO 540
       DO 545 L=POINT(K,1),POINT(K,2)
       ENDNOD=INT(COSCAP(L,1))
c  need to check if this arc has infinite cost
       IF ((ENDNOD.EQ.I).AND.(COSCAP(L,5).LT.INF)) THEN
C  AN ALTERNATE PATH HAS BEEN FOUND:
        NUMALT=NUMALT+1
570     NUMTRN=0
        PRIOR=0
        DELTA=INF
        TIS=DIST(T)
        TIA=0.0
C         WRITE(*,*)' ALT PATH FOUND:'
C ALTERNATE PATHS WILL BE A 'SPLICE' OF WHAT WE HAVE JUST FOUND
C AND THE SELECTED PATH, BASED ON HOW FAR WE HAVE TRACED BACK
        FLPATH(1)=T
        CURNODE=T
        FLNUM=1
C  THE 557 LOOP TRACES BACK ALONG THE SELECTED PATH
        DO 557 M=2,N
         IF ((M.EQ.(NUMBAK+2)).AND.(FIRST.NE.1)) THEN
          FLPATH(M)=K
         ELSE
          FLPATH(M)=PRED(CURNODE)
         ENDIF
C  FOLLOWING IF-THEN DETERMINES LAST NODE IN PATH
C  THE PRIOR.EQ.0 IF-THEN WAS ADDED IN AN ATTEMPT TO
C  TRACK TRANSSHIPMENTS PROPERLY. W/O, THE LOOP COUNTS
C  AN ADDITIONAL FAKE TRANSSHIPMENT FROM SOURCE TO ORIGIN
C  AIRBASE IF THE PATH GOES THROUGH THE TRANSSHIPMENT PATH
        IF(PRED(CURNODE).EQ.-1.0) THEN
         IF (PRIOR.EQ.0) THEN
          NUMTRN=NUMTRN-1
         ENDIF
         GOTO 565
        ENDIF
C   THE 558 LOOP ACCESSES ARC INFORMATION FROM COSCAP
        COST=-1.0
        CAPAIJ=-1.0
        FLOWIJ=-1.0
        DO 558 O=POINT(FLPATH(M),1),POINT(FLPATH(M),2)
         IF (INT(COSCAP(O,1)).EQ.CURNODE) THEN
         COST=COSCAP(O,5)
         CAPAIJ=COSCAP(O,6)
         FLOWIJ=COSCAP(O,7)
         GOTO 559
         ENDIF
```

```
558     CONTINUE
      WRITE(*,*)'>>>> ERROR, CAPAIJ NOT FOUND (DIJKSTRA).'
C CAPAIJ IS COMPARED TO PREVIOUS DELTA, MIN STORED
559     IF (CAPAIJ.GT.0.0) THEN
          D=CAPAIJ-FLOWIJ
          IF (D.LT.DELTA) DELTA=D
        ENDIF
C CURRENT ARC IS CHECKED TO SEE IF IT TRANSSHIPS
C LOGIC MODIFIED TO NOT COUNT CONSECUTIVE TRANSSHIPMENTS
C MORE THAN ONCE, PREVIOUS METHOD COUNTED
C EACH CONSECUTIVE ARC IN THE TRANSSHIPMENT PATH
C
C THE VARIABLE 'PRIOR' WILL BE THE SWITCH TO PREVENT
C NUMTRN FROM BEING INCREMENTED
C
C PRIOR='0' PREVIOUS ARC TRANSSHIPS
C PRIOR='1' PREVIOUS ARC DOESN'T TRANSSHIP
      IF (NODES(FLPATH(M),1).GT.0.0) THEN
       IF ((NODES(FLPATH(M),1).EQ.3.0).AND.
     C (NODES(CURNODE,1).NE.0.0)) THEN
        NUMTRN=NUMTRN+PRIOR
        PRIOR=0
        GOTO 561
       ELSE
        IF ((CURNODE.NE.(FLPATH(M)+1)).AND.
     C (NODES(CURNODE,1).NE.0.0)) THEN
         NUMTRN=NUMTRN+PRIOR
         PRIOR=0
         GOTO 561
        ENDIF
       ENDIF
C TIME-IN-AIR IS CALCULATED
      IF ((CURNODE.EQ.(FLPATH(M)+1)).AND.
     C (NODES(CURNODE,1).NE.0.0)) THEN
        IF (NODIKO(CURNODE,1).NE.NODIKO(FLPATH(M),1)) THEN
         TIA=TIA+COST
         PRIOR=1
        ENDIF
       ENDIF
      ENDIF
561     IF ((M.EQ.(NUMBAK+2)).AND.(FIRST.NE.1)) THEN
        CURNODE=K
        EI SE
        CURNODE=PRED(CURNODE)
        ENDIF
        FLNUM=FLNUM+1
557     CONTINUE
565   IF(FLNUM.NE.1) THEN
      WRITE(11,*)TIS,TIA,DELTA,NUMTRN,FLNUM,(FLPATH(M),M=FLNUM,1,-1)
      ENDIF
C BEST PATH LOGIC FOLLOWS
C STORE CURRENT BEST INTO BUFFER ARRAY BSTPTH(),BSTNUM,BSTDLT
```

```
      IF (DELTA.EQ.0.0) GOTO 547
      IF (NUMTRN.LT.LOTRN) GOTO 578
      IF ((NUMTRN.EQ.LOTRN).AND.(DELTA.GT.HIDELT)) GOTO 578
       GOTO 547
578     LOTRN=NUMTRN
       HIDELT=DELTA
       BSTDLT=DELTA
       BSTNUM=FLNUM
       DO 580 M=BSTNUM,1,-1
        BSTPTH(M)=FLPATH(M)
 580    CONTINUE
c       BSTPTH(1)=T
      ENDIF
547   IF (FIRST.EQ.1) THEN
      FIRST=0
      GOTO 575
     ENDIF
545   CONTINUE
540   CONTINUE
C  JUMP UP TO FIND ANOTHER ALTERNATE PATH
     GOTO 555
C  THE BEST PATH INFO NEEDS TO BE LOADED INTO PRED,DELTA:
599   IF (NUMALT.GT.0) THEN
      DO 585 M=1,BSTNUM-1
       PRED(BSTPTH(M))=BSTPTH(M+1)
585    CONTINUE
       PRED(S)=-1.0
      ENDIF
      write(11,*)
      write(11,*)
C  IF THE BEST PATH VIOLATES THE SPECIFIED MAXTRN, SET PATH=0
      IF (LOTRN.GT.MAXTRN) PATH=0
     RETURN
     END

*  SUBROUTINE COMPRESS
*  MODIFIED TO BYPASS THE OUT-AND-BACK PHENOMENON, AS WELL
*  AS RECOUNT NUMBER OF TRANSSHIPMENTS IN THE COMPRESSED PATH
*  SUBROUTINE COMPRESS TAKES A PATH DETERMINED FROM
*  DIJKSTRA AND SHORTENS IT TO A MINIMUM NUMBER OF NODES.
*  IT DELETES ANY UNNECESSARY NODES SUCH AS INTERMEDIATE
*  TRANSSHIPMENT NODES.  THE SUBROUTINE SHOULD BE CALLED
*  FOR ALL PATHS THAT ENTER THE FLOW PATTERN.
*  THE SUBROUTINE ADDS ARCS TO COSCAP AND ADJUSTS THE POINT
*  ARRAY IF COMPRESSION REQUIRES NONEXISTING ARCS.

     SUBROUTINE COMPRESS(S,T,PATH,LOTRN)

     INTEGER N,S,T,PATH
     INTEGER NUMCAR,NUMSNK,NUMMSN,ITER
     INTEGER POINT(4999,2)
     INTEGER ARCNUM,LOTRN,K
```

```
      INTEGER TRNDIS(60),TRNS(500)
      INTEGER TRNNUM,HEAD,TAIL
      INTEGER H,ADDARC,RETRN,NUMTRN
      REAL DIST(4999),PRED(4999),INF
      REAL COSCAP(89999,8),NODES(4999,4),TNODES(4999,4)
      REAL EPSILON,TISDIS(60)

      CHARACTER*4 NODIKO(4999,2)
      CHARACTER*4 CURIKO,PRIIKO

      COMMON /CONSTS/ N, INF, NUMCAR, NUMSNK, NUMMSN, ITER,EPSILON
      COMMON /ARRS/ COSCAP, POINT, NODES, NODIKO, TNODES
      COMMON /FLOW/ PRED, DIST, ARCNUM, TRNDIS, TRNS, TISDIS

*  CHECK CONDITIONS TO JUSTIFY RUNNING COMPRESS
      IF (PATH.EQ.0) THEN
       WRITE(*,*)' ERROR, COMPRESS CALLED, STPATH=0'
       GOTO 3120
      ENDIF
*  ADDITIONAL LOOP ADDED TO CATCH OUT-AND-BACKS
*  RETRN IS A SWITCH TO INDICATE IF ANY ARCS HAVE BEEN ADDED
*  IF THEY HAVE, RETRN SET TO 1, AND THE NUMBER OF TRANSSHIPMENTS
*  MUST BE RETALLIED.
      RETRN=0
      H=T
3070  H=PRED(H)
      IF ((PRED(H).EQ.S).OR.(H.EQ.S)) GOTO 3080
      I=H
      HEAD=I
      PRIIKO=NODIKO(I,1)
      TRNNUM=0
      ADDARC=0
3000  IF(I.EQ.S) GOTO 3010
      IF(I.EQ.-1) THEN
        WRITE(*,*)'ERROR:  I = -1! (COMPRESS)'
        WRITE(*,*)'PRED ARRAY INCORRECT'
      ENDIF
      J=I
      I=PRED(J)
      CURIKO=NODIKO(I,1)
      TRNNUM=TRNNUM+1
      IF ((CURIKO.EQ.PRIIKO).AND.(TRNNUM.GE.2)) THEN
       ADDARC=1
       RETRN=1
       TAIL=I
       PRED(HEAD)=TAIL
      ENDIF
*  ERROR CHECKER:
      IF (TRNNUM.GT.N) THEN
       WRITE(*,*)'  INFINITE LOOP IN COMPRESS, ABORT'
       WRITE(*,*)'  S=',S,' T=',T
       WRITE(*,*)'  I=',I,' J=',J
```

```
      STOP
      ENDIF
      GOTO 3000
*  IF ADDARC=1, AN ARC HAS BEEN INTRODUCED,
*  NEED TO SCAN TO SEE IF IT EXISTS IN COSCAP,
*  IF IT DOESN'T, INSERT AND UPDATE COSCAP,POINT
3010  IF (ADDARC.EQ.1) THEN
*  SEARCH FOR THE ARC IN COSCAP
        DO 3020 K=POINT(TAIL,1),POINT(TAIL,2)
        IF (INT(COSCAP(K,1)).EQ.HEAD) GOTO 3070
3020    CONTINUE
*  SHIFT ALL FOLLOWING ARCS ONE DOWN IN COSCAP
      ARCNUM=ARCNUM+1
      DO 3040 K=ARCNUM,POINT(TAIL,2)+2,-1
       COSCAP(K,1)=COSCAP(K-1,1)
       COSCAP(K,2)=COSCAP(K-1,2)
       COSCAP(K,3)=COSCAP(K-1,3)
       COSCAP(K,4)=COSCAP(K-1,4)
       COSCAP(K,5)=COSCAP(K-1,5)
       COSCAP(K,6)=COSCAP(K-1,6)
       COSCAP(K,7)=COSCAP(K-1,7)
       COSCAP(K,8)=COSCAP(K-1,8)
3040  CONTINUE
*  INSTALL NEW ARC INTO COSCAP
      K=POINT(TAIL,2)+1
      COSCAP(K,1)=HEAD
      COSCAP(K,2)=NODES(HEAD,2)-NODES(TAIL,2)
      COSCAP(K,3)=INF
      COSCAP(K,4)=0.0
      COSCAP(K,5)=COSCAP(K,2)
      COSCAP(K,6)=COSCAP(K,3)
      COSCAP(K,7)=COSCAP(K,4)
      COSCAP(K,8)=NODES(HEAD,4)
*  UPDATE ENDPOINTER FOR NODE TAIL
      POINT(TAIL,2)=K
*  SHIFT ALL SUBSEQUENT NODE POINTERS BY 1 (EXCEPT -1'S)
      DO 3060 K=TAIL+1,N
       IF (POINT(K,1).GT.0) THEN
        POINT(K,1)=POINT(K,1)+1
        POINT(K,2)=POINT(K,2)+1
       ENDIF
3060  CONTINUE
      ENDIF
*  ENDIF FOR (ADDARC.EQ.1) ^^^
*  LOOP BACK TO CONTINUE CHECKING FOR OUT-AND-BACKS
      GOTO 3070
*  ADD CODE TO COUNT TRANSSHIPMENTS OF COMPRESSED PATH
3080  IF (RETRN.EQ.0) GOTO 3120
      NUMTRN=0
      I=PRED(T)
3090  IF(PRED(I).EQ.S) GOTO 3100
      J=I
```

```
      I=PRED(J)
*  MSN NUMBER DIFFERENCE CONSIDERED A TRANSSHIPMENT
      IF (I.NE.(J-1)) THEN
       NUMTRN=NUMTRN+1
      ENDIF
      GOTO 3090
3100  LOTRN=NUMTRN
3120  RETURN
      END

*     SUBROUTINE POSTPROC PERFORMS POST-PROCESSING ON THE OUTPUT FROM
*     SUBROUTINE CARGFLO
*     IT DETERMINES THE UTILIZATION OF THE LEGS ALONG EACH MISSION,
*     REPORTING THIS UTILIZATION AS A PERCENTAGE OF TOTAL CAPACITY USED.

      SUBROUTINE POSTPROC ()

      INTEGER N,PATHNUM,PATH(500,75),NUMCAR
      INTEGER NUMSNK,MSNNOD,NUMNODE(4999),BEGMSN(1000),ENDMSN(1000)
      INTEGER FLONUM(4999),NUM(4999),NUMMSN
      INTEGER ITER,TEMP,POINT(4999,2),ARCNUM
      INTEGER MSNORD(1000),TMPORD,ORDER
      INTEGER TRNDIS(60),TRNS(500),SORCRI,TMPORD,UPPER

      REAL COSCAP(89999,8),INF,PRED(4999),DIST(4999),UTIL
      REAL NODES(4999,4),TNODES(4999,4),ACCAP(1000),RATIO
      REAL PTHFLO(500),MAXUTE(1000),MAXUTL
      REAL EPSILON,TISDIS(60)
      REAL MSNCAP,TIMEPS, TMPUTE

      CHARACTER PPNAM*9,NS*3,EXT*2,POS(3)*1,NODIKO(4999,2)*4
      CHARACTER AC(1000)*4,MSNCOD(1000)*1

      COMMON /CONSTS/ N, INF, NUMCAR, NUMSNK, NUMMSN, ITER,EPSILON
      COMMON /PARAMS/ MAXIT,MAXALT,MAXTRN,CARCRI,PASSES,SORCRI,TIMEPS
      COMMON /ARRS/ COSCAP, POINT, NODES, NODIKO, TNODES
      COMMON /FLOW/ PRED, DIST, ARCNUM, TRNDIS, TRNS, TISDIS
      COMMON /MARK/ NUMNODE, AC, ACCAP, BEGMSN, ENDMSN, NUM, PATH,
     C        FLONUM, PATHNUM, PTHFLO, MSNCOD
      COMMON /ORD/ MSNORD

      OPEN(UNIT=2,FILE='newsched.dat',STATUS='OLD',FORM='FORMATTED')
*     THE FOLLOWING SECTION DETERMINES THE NODE NUMBERS ASSOCIATED WITH A
*     PARTICULAR MISSION.  BEGMSN(I) IS THE FIRST NODE ON MISSION I AND
*     ENDMSN(I) IS THE FINAL NODE.
      MSNNOD=NUMCAR+1
      READ(2,*)NUMMSN
      DO 1000 I=1,NUMMSN
        READ(2,*)NUMNODE(I),AC(I),ACCAP(I),MSNCOD(I)
        BEGMSN(I)=MSNNOD
        ENDMSN(I)=BEGMSN(I)+NUMNODE(I)-1
        NUM(I)=NUMNODE(I)
```

```
      DO 1010 J=1,NUMNODE(I)
        READ(2,*)
1010    CONTINUE
      MSNNOD=MSNNOD+NUMNODE(I)
1000  CONTINUE
*     THE FOLLOWING LOOP DESIGNATES A UNIQUE FILENAME TO EACH ITERATION'S
*     POST-PROCESSING.
      TEMP=ITER
      EXT='.c'
      DO 1020 I=2,0,-1
       RATIO=INT(TEMP/(10**I))
       TEMP=TEMP-RATIO*(10**I)
       POS(I+1)=CHAR(48+RATIO)
1020  CONTINUE
      NS=POS(3)//POS(2)//POS(1)
      PPNAM='post'//NS//EXT
      OPEN(UNIT=3,FILE=PPNAM,STATUS='UNKNOWN',FORM='FORMATTED')
*     BEGIN POST-PROCESSING
*     WRITE HEADER
      WRITE(3,*)'UTILIZATION OF MISSIONS'
      WRITE(3,*)
      WRITE(3,*)'UTILIZATION EQUALS THE PERCENTAGE OF A MISSION LEG '
      WRITE(3,*)'CAPACITY THAT IS USED.'
      WRITE(3,*)
      WRITE(3,*)'ICAO  UTIL   FLOW'
      WRITE(3,*)'____ ____  ____'
      WRITE(3,*)
      DO 1030 I=1,NUMMSN
        MAXUTE(I)=0.0
        MSNCAP=0.0
      WRITE(3,1040)I,AC(I),ACCAP(I)
1040  FORMAT('MISSION ',I3,' (ACFT = ',A4,', CAPACITY = ',
     c       F4.1,' TONS)')
      WRITE(3,1050)NODIKO(BEGMSN(I),1)
1050  FORMAT(A4,2X,'----  ----')
        DO 1060 J=BEGMSN(I)+1,ENDMSN(I)
          IF(POINT(J-1,1).EQ.-1) THEN
            WRITE(*,*)'BEFORE 1070, -1'
          ENDIF
          DO 1070 K=POINT(J-1,1),POINT(J-1,2)
            IF(INT(COSCAP(K,1)).EQ.J) THEN
              IF(NODIKO(J-1,1).NE.NODIKO(J,1)) THEN
                UTIL=COSCAP(K,7)/COSCAP(K,3)
                MAXUTE(I)=MAXUTE(I)+COSCAP(K,2)*COSCAP(K,7)
                MSNCAP=MSNCAP+(COSCAP(K,2)*COSCAP(K,3))
                  WRITE(3,1080)NODIKO(J,1),UTIL,COSCAP(K,7)
1080                FORMAT(A4,2X,F4.2,2X,F6.2)
              ELSE
                GOTO 1070
              ENDIF
            ENDIF
1070      CONTINUE
```

```
1060   CONTINUE
       MAXUTE(I)=MAXUTE(I)/MSNCAP
       WRITE(3,*)'OVERALL UTILIZATION ON THIS MISSION: ',MAXUTE(I)
       IF(MAXUTE(I).GT.1.0) THEN
         WRITE(*,*)'WARNING:  MISSION ',I,' IS OVERUTILIZED!'
       ENDIF
       WRITE(3,*)
1030  CONTINUE
*      THIS SECTION ORDERS THE MISSION SET ACCORDING TO USER PREFERENCE
       DO 1090 I=1,NUMMSN
         MSNORD(I)=I
1090  CONTINUE
       IF(SORCRI.EQ.1) GOTO 1105
       IF(SORCRI.EQ.2) THEN
*      THIS SECTION SORTS IN REVERSE GIVEN ORDER
         DO 1095 I=1,NUMMSN
           MSNORD(I)=NUMMSN+1-I
1095     CONTINUE
         GOTO 1105
       ENDIF
       IF(SORCRI.GT.2) THEN
*        THIS SECTION SORTS ON ASCENDING UTILIZATION
         DO 1100 I=1,NUMMSN-1
           MAXUTL=MAXUTE(I)
           ORDER=I
           DO 1110 J=I+1,NUMMSN
             IF(MAXUTE(J).GT.MAXUTL) THEN
               MAXUTL=MAXUTE(J)
               ORDER=J
             ENDIF
1110       CONTINUE
*          SWITCH POSITION I AND POSITION ORDER IN MSNORD ARRAY
           TMPUTE=MAXUTE(I)
           MAXUTE(I)=MAXUTE(ORDER)
           MAXUTE(ORDER)=TMPUTE
           TMPORD=MSNORD(I)
           MSNORD(I)=MSNORD(ORDER)
           MSNORD(ORDER)=TMPORD
1100    CONTINUE
       ENDIF
       IF(SORCRI.EQ.4) THEN
*      THIS SECTION SORTS ON DESCENDING UTILIZATION (REVERSES THE ORDER
*      DETERMINED BY THE ABOVE 1100 LOOP
*      SINCE WE ARE DOING A PAIRWISE SWITCH, WE NEED ONLY GO THROUGH THE
*      FIRST HALF OF THE MISSIONS.  WE MUST DETERMINE IF THE NUMBER OF
*      MISSIONS IS ODD OR EVEN.
       IF((NUMMSN/2.).GT.(INT(NUMMSN/2.))) THEN
*        ODD NUMBER
         UPPER=INT(NUMMSN/2.)+1
       ELSE
         UPPER=INT(NUMMSN/2.)
       ENDIF
```

```
      DO 1096 I=1,UPPER
        TMPORD=MSNORD(I)
        MSNORD(I)=MSNORD(NUMMSN-I+1)
        MSNORD(NUMMSN-I+1)=TMPORD
1096    CONTINUE
      ENDIF
*     THE MISSION SET IS NOW SORTED ACCORDING TO USER PREFERENCE
1105  CLOSE(2)
      CLOSE(3)
      RETURN
      END

*     THIS SUBROUTINE DOES SOME PRE-PROCESSING OF THE DATA FOR LATER USE
*     IN SUBROUTINE MODMSN.

      SUBROUTINE PREMOD()

      INTEGER NUMCAR,NUMMSN,NUMNODE(4999),BEGMSN(1000)
      INTEGER ENDMSN(1000),NUM(4999),ARCNUM,POINT(4999,2)
      INTEGER PATHNUM,PATH(500,75),FLONUM(4999)
      INTEGER TRNDIS(60),TRNS(500)

      REAL COSCAP(89999,8),ACCAP(1000),NODES(4999,4),PRED(4999)
      REAL DIST(4999),TISYS,TNODES(4999,4),PTHFLO(500)
      REAL EPSILON,TISDIS(60)

      CHARACTER*4 AC(1000),NODIKO(4999,2),MSNCOD(1000)*1

      COMMON /CONSTS/ N, INF, NUMCAR, NUMSNK, NUMMSN, ITER,EPSILON
      COMMON /ARRS/ COSCAP, POINT, NODES, NODIKO, TNODES
      COMMON /FLOW/ PRED, DIST, ARCNUM, TRNDIS, TRNS, TISDIS
      COMMON /MARK/ NUMNODE, AC, ACCAP, BEGMSN, ENDMSN, NUM, PATH,
     C         FLONUM, PATHNUM, PTHFLO, MSNCOD

      OPEN(UNIT=4,FILE='paths.out',STATUS='OLD',FORM='FORMATTED')

*     THIS SECTION READS IN AND STORES THE PATHS GENERATED BY SUBROUTINE
*     CARGFLO.  IT ALSO STORES THE TIME IN SYSTEM AND FLOW FOR EACH PATH.
      PATHNUM=0
      DO 1120 I=1,99999999
        READ(4,*,END=1130)TISYS,TRNS(I),PTHFLO(I),FLONUM(I),
     C                    (PATH(I,J),J=1,FLONUM(I))
        PATHNUM=PATHNUM+1
1120  CONTINUE
1130  CONTINUE
*     THE FOLLOWING LOOP SETS COSCAP(I,8) TO ZERO FOR ALL ARCS.
      DO 1140 I=1,ARCNUM
        COSCAP(I,8)=0.
1140  CONTINUE
*     THIS SECTION MARKS COLUMN 8 OF COSCAP (I.E. THE MISSION THAT THE
*     ARC CONNECTS TO) FOR EVERY ARC IN THE FLOW.  THAT IS, ONLY THOSE
*     ARCS THAT ARE ACTUALLY USED (NON-ZERO FLOW) WILL BE MARKED.
```

```
      DO 1170 I=1,PATHNUM
        DO 1180 J=1,FLONUM(I)-2
          DO 1190 K=POINT(PATH(I,J),1),POINT(PATH(I,J),2)
            IF(INT(COSCAP(K,1)).EQ.PATH(I,J+1)) THEN
              COSCAP(K,8)=NODES(PATH(I,J+1),4)
            ENDIF
1190      CONTINUE
1180    CONTINUE
1170  CONTINUE
*     THE FOLLOWING SECTION STORES THE VALUES OF THE NODES MATRIX IN THE
*     TEMPORARY MATRIX TNODES.
      DO 1210 I=1,N
        DO 1220 J=1,4
          TNODES(I,J)=NODES(I,J)
1220    CONTINUE
1210  CONTINUE
*     THE FOLLOWING SECTION STORES THE COSTS ASSOCIATED WITH EACH ARC IN
*     THE TEMPORARY COLUMN OF THE COSCAP MATRIX.
      DO 1230 I=1,ARCNUM
        COSCAP(I,5)=COSCAP(I,2)
1230  CONTINUE
      CLOSE(4)
      RETURN
      END

*     SUBROUTINE MODMSN COMBINED WITH SUBROUTINE STEP3, IMPLEMENTS THE
*     SCHEDULING IMPROVEMENT ALGORITHM, WHICH IS DESIGNED TO SHIFT THE
*     START TIMES OF THE MISSIONS TO AN EARLIER TIME (SHIFTING TO A LATER
*     TIME WILL HAVE TO BE A TOPIC OF FUTURE RESEARCH) IN AN ATTEMPT TO
*     DELIVER CARGO SOONER TO THE CUSTOMER AND THUS REDUCING THE OVERALL
*     COST.
*     THERE ARE FOUR MAIN STEPS IN THE ALGORITHM:
*       STEP 1:  DETERMINATION OF THE TIME SHIFT
*       STEP 2:  IMPLEMENTATION OF THE TIME SHIFT
*       STEP 3:  MEASURING THE IMPACT OF THE TIME SHIFT
*       STEP 4:  REVERSAL OF THE TIME SHIFT
*     SUBROUTINE MODMSN PERFORMS STEPS 1, 2, AND 4, WHILE THE APTLY NAMED
*     SUBROUTINE STEP3 PERFORMS STEP 3.
*     PREVIOUS WORK, SPECIFICALLY THE RAU THESIS PROJECT, SHIFTED THE
*     MISSION START TIMES ONLY TO THE POINT OF MAINTAINING THE CURRENT
*     FLOW.  THIS ALGORITHM IMPROVES UPON THIS BY ALLOWING CARGO TO BE
*     REFLOWED (FLOWED ALONG DIFFERENT PATHS) IF IT GENERATES A BETTER
*     FLOW PATTERN.  A BETTER FLOW PATTERN IS DEFINED AS ONE WHICH DELIVERS
*     AT LEAST AS MUCH CARGO AS BEFORE WITH A SMALLER OVERALL COST.  IF
*     ANY PREVIOUSLY DELIVERED CARGO CANNOT BE DELIVERED BECAUSE OF A CHANGE
*     IN THE SCHEDULE, THE CHANGE IS NOT IMPLEMENTED.
*************************************************************************
      SUBROUTINE MODMSN(MODTOT,MODFLO,TOTAL,TERCRI)

      INTEGER N,K,PATHNUM,PATH(500,75),NUMCAR
      INTEGER NUMSNK,NUMNODE(4999),BEGMSN(1000),ENDMSN(1000)
      INTEGER FLONUM(4999),CHANGE,NUM(4999),NUMMSN
```

```
      INTEGER ITER,TEMP,POINT(4999,2),ARCNUM
      INTEGER NUMPATH,FEAS
      INTEGER ARCCNT,CALLS,TFLONM(500),TPATH(500,75)
      INTEGER MSNORD(1000)
      INTEGER MAXIT,MAXALT,MAXTRN,CARCRI,PASSES,MSNPAS,SORCRI
      INTEGER TRNDIS(60),TRNS(500),TTRNS(500)
      INTEGER BEGARC(500),ENDARC(500),TRNCNT
      INTEGER FLNUM,INPATH(75),TRANS,TERCRI

      REAL COSCAP(89999,8),INF,TCOST,PRED(4999),DIST(4999)
      REAL NODES(4999,4),TNODES(4999,4),ACCAP(1000),SHIFT,RATIO
      REAL COST(2),PTHFLO(500),TOTAL,TISDIS(60)
      REAL MODTOT,MODFLO,TISYS,FLO,REFLOW,TIMEPS,EPSILON

      CHARACTER FILNAM*10,NS*3,EXT*2,POS(3)*1,NODIKO(4999,2)*4
      CHARACTER AC(1000)*4,PTHNAM*10,MSNCOD(1000)*1

      COMMON /CONSTS/ N, INF, NUMCAR, NUMSNK, NUMMSN, ITER,EPSILON
      COMMON /PARAMS/ MAXIT,MAXALT,MAXTRN,CARCRI,PASSES,SORCRI,TIMEPS
      COMMON /ARRS/ COSCAP, POINT, NODES, NODIKO, TNODES
      COMMON /FLOW/ PRED, DIST, ARCNUM, TRNDIS, TRNS, TISDIS
      COMMON /MARK/ NUMNODE, AC, ACCAP, BEGMSN, ENDMSN, NUM, PATH,
     C        FLONUM, PATHNUM, PTHFLO, MSNCOD
      COMMON /CHNG/ NUMPATH, BEGARC, ENDARC
      COMMON /ORD/ MSNORD

*     - N IS THE NUMBER OF NODES IN THE NETWORK
*     - PATHNUM IS A COUNTER FOR THE NUMBER OF PATHS USED IN THE FLOW.
*     - PATH(I,J) IS A MATRIX OF NODE IDENTIFIERS.  FOR EXAMPLE, PATH(1,3)
*       IS THE THIRD NODE ON THE FIRST PATH.
*     - NODENUM(I) IS THE NUMBER OF NODE I.
*     - NUMCAR IS THE NUMBER OF CARGO GENERATION NODES.
*     - NUMSNK IS THE NUMBER OF SINK NODES.
*     - MSNNOD IS USED TO STORE THE NUMBER OF THE BEGINNING NODE FOR A
*       PARTICULAR MISSION.
*     - NUMNODE(I) IS THE NUMBER OF NODES ASSOCIATED WITH MISSION I.
*     - BEGMSN(I) AND ENDMSN(I) ARE ARRAYS WHICH HOLD THE NUMBERS OF THE
*       BEGINNING AND END NODES FOR MISSION I.
*     - NUMMSN IS THE NUMBER OF MISSIONS ON THE NETWORK.
*     - NUM(I) IS AN ARRAY WHICH HOLDS THE VALUE OF NUMNODE FOR A MISSION I.
*     - FLONUM(I) IS AN ARRAY HOLDING THE NUMBER OF NODES USED ON PATH I.
*     - CHANGE IS A COUNTER FOR THE NUMBER OF SCHEDULE CHANGES MADE DURING A
*       SINGLE PASS THROUGH THE NETWORK.
*     - NODES(I,3) IS A MATRIX CONTAINING BASE ID, TIME, AND CARGO FOR NODE I.
*     - TNODES(I,3) IS A TEMPORARY MATRIX CONTAINING THE SAME INFORMATION HELD
*       IN NODES(I,3).
*     - SHIFT IS THE CHANGE IN THE SCHEDULE.
*     - INF IS INFINITIY.
*     - COSCAP(I,7) IS A MATRIX CONTAINING THE END NODE, COST, AND CAPACITY
*       FOR ARC I.
*     - POINT(I,2) IS A MATRIX HOLDING THE ARC NUMBERS EMANATING FROM NODE I.
*     - TCOST IS A TEMPORARY VARIABLE CONTAINING THE PROPOSED TIME SHIFT.
```

```
*   THE OUTPUT FILE 'modmsn.c' HOLDS A LOG OF THE ACTIONS TAKEN BY THE
*   PROGRAM.  IT IS USED PRIMARILY FOR DIAGNOSTIC PURPOSES AND MAY BE
*   'TURNED OFF' WITHOUT AFFECTING THE PROGRAMMING.  JUST REMEMBER TO
*   COMMENT OUT ALL THE APPROPRIATE WRITE STATEMENTS.
*************************************************************************
*   BEGIN THE SCHEDULING IMPROVEMENT ALGORITHM                          *
*************************************************************************
*   STEP 0 -- INITIALIZATION
    MSNPAS=0
    TERCRI=0
1240  CHANGE=0
    CALLS=0
    MSNPAS=MSNPAS+1
*************************************************************************
*   STEP 1:  DETERMINATION OF THE TIME SHIFT                            *
*************************************************************************
*   GENERAL DESCRIPTION OF STEP 1:
*   FOR MISSION M FIND ALL THE TRANSSHIPMENT ARCS THAT TERMINATE ON THE
*   MISSION.  DETERMINE WHICH ARC WILL BE THE FIRST TO BECOME INFEASIBLE
*   WITH A TIME SHIFT.  WE CALL THIS ARC THE MOST CRITICAL ARC AND STORE
*   ITS COST AS COST(1).  ALSO FIND THE NEXT ARC WHICH WILL BECOME
*   INFEASIBLE AND CALL IT THE NEXT MOST CRITICAL ARC, STORING ITS COST
*   AS COST(2).
*   SINCE WE CANNOT SHIFT A MISSION BY SUCH AN AMOUNT THAT WILL CAUSE ITS
*   START TIME TO BE A NEGATIVE NUMBER, WE DEFINE THE TIME SHIFT TO BE
*   THE MINIMUM OF {THE MISSION START TIME, COST(2)}.  THIS TIME SHIFT
*   REPRESENTS THE MAXIMUM AMOUNT OF TIME WE CAN SHIFT THE MISSION START
*   WHILE ONLY CAUSING THE MOST CRITICAL ARC TO BECOME INFEASIBLE.
    DO 1250 A=1,NUMMSN
     K=MSNORD(A)
*   IF THE MISSION CODE FOR THIS MISSION IS 'F', THIS MISSION IS A FREQUENCY
*   REQUIREMENT MISSION.  THIS ALGORITHM DOES NOT ALTER FREQUENCY
*   MISSIONS, SO WE MOVE ON TO THE NEXT MISSION.
    IF(MSNCOD(K).EQ.'F') GOTO 1250
    FEAS=0
    SHIFT=INF
    COST(1)=INF
    ARCCNT=0
*   FIND ALL TRANSSHIPMENT ARCS THAT TERMINATE ON THE MISSION
     DO 1260 L=1,ARCNUM
      IF(INT(COSCAP(L,8)).EQ.K) THEN
*IF(1)
        DO 1270 M=1,N
         IF(L.GE.POINT(M,1).AND.L.LE.POINT(M,2)) THEN
*IF(2)
*   DISREGARD SEQUENTIAL ARCS IF ON THE SAME MISSION
           IF(NODES(M,4).EQ.K) GOTO 1260
           ARCCNT=ARCCNT+1
           TCOST=COSCAP(L,5)
*   IF THE TCOST IS LESS THAN THE CURRENT VALUE OF SHIFT, MAKE TCOST THE
*   NEW VALUE OF SHIFT AND ADJUST THE TWO COSTS.
           IF(TCOST.LT.SHIFT) THEN
```

```
*IF(3)
                SHIFT=TCOST
                COST(2)=COST(1)
                COST(1)=SHIFT
              ELSE
*ELSE(3)
              IF(TCOST.GT.SHIFT) THEN
*   IF TCOST IS GREATER THAN THE CURRENT VALUE OF SHIFT BUT LESS THAN
*   THE VALUE OF COST(2), SET COST(2) EQUAL TO TCOST.
                IF(TCOST.LT.COST(2)) COST(2)=TCOST
              ENDIF
              ENDIF
*ENDIF(3)
            ENDIF
*ENDIF(2)
1270        CONTINUE
         ENDIF
*ENDIF(1)
1260    CONTINUE
*   IF NO TRANSSHIPMENT ARCS TERMINATED ON THE MISSION, A SHIFT OF THE
*   MISSION WILL NOT CAUSE ANY INFEASIBILITIES. DISREGARD THIS MISSION
*   AND PROCEED TO THE NEXT.
    IF(ARCCNT.EQ.0) GOTO 1250
*   THE MISSION CAN ONLY BE SHIFTED BY THE MINIMUM OF THE MISSION
*   STARTING TIME AND THE SHIFT DETERMINED ABOVE.
         SHIFT=COST(2)
*   THERE MAY BE MORE THAN 1 MOST CRITICAL ARC. THAT IS, THERE MAY
*   BE MULTIPLE ARCS WITH SAME COST THAT WILL BECOME INFEASIBLE WITH A
*   TIME SHIFT. THIS SECTION FINDS ALL OF THEM, DISREGARDING THOSE
*   THAT FALL ON THE SAME MISSION. AN ARC CONNECTING TWO NODES ON THE
*   SAME MISSION WILL NEVER BECOME INFEASIBLE WITH A TIME SHIFT.
    TRNCNT=0
    DO 1265 L=1,ARCNUM
     IF(INT(COSCAP(L,8)).NE.K) GOTO 1265
     IF(COSCAP(L,5).EQ.COST(1)) THEN
*   FIND THE BEGINNING NODE ASSOCIATED WITH THIS ARC.
       DO 1275 M=1,N-NUMSNK
         IF(L.GE.POINT(M,1).AND.L.LE.POINT(M,2)) THEN
*   NODE M IS THE BEGINNING NODE OF THE ARC
           IF(INT(NODES(M,4)).EQ.K) GOTO 1265
           TRNCNT=TRNCNT+1
           BEGARC(TRNCNT)=M
           ENDARC(TRNCNT)=INT(COSCAP(L,1))
           GOTO 1265
         ENDIF
1275     CONTINUE
     ENDIF
1265  CONTINUE
*   IF ONLY 1 TRANSSHIPMENT ARC TERMINATED ON THE MISSION, THE VALUE
*   OF SHIFT WILL BE INFINITY. SINCE WE OBVIOUSLY CANNOT SHIFT BY
*   THIS AMOUNT, WE MUST ARBITRARILY DEFINE A VALUE FOR IT. WE NEED A
*   VALUE GREATER THAN COST(1), BUT NOT TOO MUCH GREATER, SINCE WE HOPE
```

```
*   TO PERTURB THE SYSTEM BY AS LITTLE AS POSSIBLE.  ARBITRARILY SET
*   SHIFT TO COST(1)+.1.
        IF(SHIFT.EQ.INF) SHIFT=COST(1)+.1
        IF(NODES(BEGMSN(K),2).LE.SHIFT) THEN
*IF(4)
          SHIFT=NODES(BEGMSN(K),2)
          IF(NODES(BEGMSN(K),2).LE.COST(1)) THEN
*IF(5)
            DO 1266 Q=1,TRNCNT
              BEGARC(Q)=0
              ENDARC(Q)=0
1266           CONTINUE
          ENDIF
*ENDIF(5)
        ENDIF
*ENDIF(4)
   IF(SHIFT.LT.TIMEPS) THEN
     SHIFT=SHIFT+TIMEPS
   ENDIF
   IF((SHIFT-TIMEPS).LE.TIMEPS) THEN
     SHIFT=0.
   ELSE
     SHIFT=SHIFT-TIMEPS
   ENDIF
*   THE TIME SHIFT HAS NOW BEEN DETERMINED.
*********************************************************************************
*   STEP 2:  IMPLEMENTATION OF THE TIME SHIFT                    *
*********************************************************************************
*   GENERAL DESCRIPTION OF STEP 2:
*   THE IMPLEMENTATION OF THE MISSION'S TIME SHIFT IS PERFORMED BY SUB-
*   TRACTING THE AMOUNT OF THE TIME SHIFT FROM THE TIMES ASSOCIATED
*   WITH EACH NODE ALONG THE MISSION.
*   IF IN STEP 1 WE DETERMINED THAT THE TIME SHIFT WAS 0.0, THEN THE
*   NETWORK WILL NOT BE CHANGED AND WE CAN RETURN TO STEP 1 WITH A NEW
*   MISSION.
*   OTHERWISE, THE STATE OF THE NETWORK HAS BEEN CHANGED AND THE COSTS
*   OF THE TRANSSHIPMENT ARCS MUST BE UPDATED TO REFLECT THE TIME SHIFT.
*   NOT ALL OF THE TRANSSHIPMENT ARCS HAVE CHANGED, HOWEVER.  ONLY THOSE
*   TRANSSHIPMENT ARCS WHICH ORIGINATE OR TERMINATE ON THE MISSION NEED
*   TO BE UPDATED.
*   IT IS RECOGNIZED THAT A TIME SHIFT MAY CAUSE THE OVERALL COST OF THE
*   FLOW TO INCREASE, OR MAY CAUSE SOME PREVIOUSLY FLOWED CARGO TO REMAIN
*   UNFLOWED.  SINCE WE CANNOT KNOW THIS BEFORE PERFORMING STEP 2, WE WILL
*   STORE THE CURRENT STATE OF THE NETWORK PRIOR TO PERFORMING STEP 3 IN
*   CASE THE CONDITIONS IN STEP 3 ARE NOT SATISFIED.  IF THESE CONDITIONS
*   ARE NOT SATISFIED, WE WILL RESTORE THE PRE-SHIFT STATE OF THE NETWORK.
*         IF THE SHIFT EQUALS THE MISSION START TIME, SHIFT THE MISSION
*         AND UPDATE NODES AND COSCAP APPROPRIATELY.  NO ARCS HAVE
*         BECOME INFEASIBLE.  INCREMENT CHANGE BY 1.
*         IF THE SHIFT EQUALS COST(2) (DEFINED ABOVE), THEN TEMPORARILY
*         SHIFT THE MISSION AND UPDATE TNODES AND COSCAP APPROPRIATELY.
*         INCREMENT CHANGE BY 1.
```

```
*    IF THE SHIFT EQUALS 0 OR INFINITY, THERE IS NO NEED TO PERFORM A SHIFT.
     IF(SHIFT.NE.INF.AND.SHIFT.GT.0.) THEN
*IF(6)
*    STORE PATHS, CAPACITIES, AND NODE TIMES (FOR THIS MISSION) IN CASE
*    CHANGES HAVE TO BE UNDONE LATER.
     DO 1280 M=1,PATHNUM
       TTRNS(M)=TRNS(M)
       TFLONM(M)=FLONUM(M)
       DO 1290 V=1,FLONUM(M)
         TPATH(M,V)=PATH(M,V)
1290   CONTINUE
1280  CONTINUE
      DO 1300 V=BEGMSN(K),ENDMSN(K)
        TNODES(V,2)=NODES(V,2)
1300   CONTINUE
      DO 1315 V=1,ARCNUM
        COSCAP(V,2)=COSCAP(V,8)
        COSCAP(V,3)=COSCAP(V,6)
        COSCAP(V,4)=COSCAP(V,7)
1315   CONTINUE
      DO 1310 V=BEGMSN(K),ENDMSN(K)
        NODES(V,2)=NODES(V,2)-SHIFT
1310   CONTINUE
*    SETTING FEAS EQUAL TO 1 INDICATES THAT A FEASIBLE SHIFT WILL OCCUR.
*    THAT IS, NO CARGO WILL HAVE TO BE REFLOWED.
      IF(BEGARC(1).EQ.0) FEAS=1
*    NOW THAT THE TIMES ASSOCIATED WITH THE MISSION HAVE BEEN
*    CHANGED, THE TIMES IN THE NODES MATRIX AND THE COSTS IN THE
*    COSCAP MATRIX MUST BE UPDATED.
*    CHANGE COSCAP FOR EVERY ARC THAT TERMINATES ON THE MISSION
1320  DO 1330 W=1,N-NUMSNK
      IF(POINT(W,1).LT.0) GOTO 1330
      DO 1360 V=POINT(W,1),POINT(W,2)
        IF(INT(NODES(INT(COSCAP(V,1)),4)).EQ.K) THEN
*IF(7)
          IF(NODES(W,2).LE.NODES(INT(COSCAP(V,1)),2)) THEN
*IF(9)
            COSCAP(V,5)=NODES(INT(COSCAP(V,1)),2)-
   C                    NODES(W,2)
            IF(ABS(COSCAP(V,6)-COSCAP(V,7)).LE.EPSILON) THEN
*IF(9.5)
              COSCAP(V,5)=INF
            ENDIF
*ENDIF(9.5)
          ELSE
*ELSE(9)
            COSCAP(V,5)=INF
          ENDIF
*ENDIF(9)
        ENDIF
1360   CONTINUE
*ENDIF(7)
```

```
1330 CONTINUE
*    WE MUST ALSO CHANGE COSCAP FOR ALL THE ARCS EMANATING FROM THE MISSION
     DO 1370 W=BEGMSN(K),ENDMSN(K)
        IF (POINT(W,1).LT.0) GOTO 1370
       DO 1380 V=POINT(W,1),POINT(W,2)
        IF(INT(COSCAP(V,1)).GT.N-NUMSNK) GOTO 1380
        IF(NODES(W,2).LE.NODES(INT(COSCAP(V,1)),2)) THEN
*IF(10)
          COSCAP(V,5)=NODES(INT(COSCAP(V,1)),2)-NODES(W,2)
           IF(ABS(COSCAP(V,6)-COSCAP(V,7)).LE.EPSILON) THEN
*IF(10.6)
             COSCAP(V,5)=INF
           ENDIF
*ENDIF(10.6)
        ELSE
*ELSE(10)
          COSCAP(V,5)=INF
        ENDIF
*ENDIF(10)
1380    CONTINUE
1370 CONTINUE
     CONTINUE
*    THE TIME SHIFT HAS NOW BEEN IMPLEMENTED.  IF ONLY A FEASIBLE SHIFT IS
*    APPLICABLE, WE NEED NOT PROCEED TO STEP 3.  INSTEAD, WE RETURN TO
*    STEP 1 WITH THE NEXT MISSION.
     IF(FEAS.EQ.1) THEN
*IF(10.5)
*    COMPUTE THE NEW OVERALL COST OF THE FLOW.
1375    TOTAL=0.
       DO 1410 M=1,PATHNUM
        TIS=NODES(PATH(M,FLONUM(M)-1),2)-NODES(PATH(M,1),2)
        TOTAL=TOTAL+(TIS*PTHFLO(M))
1410    CONTINUE
       DO 1415 M=1,ARCNUM
        COSCAP(M,6)=COSCAP(M,3)
        COSCAP(M,7)=COSCAP(M,4)
        COSCAP(M,8)=COSCAP(M,2)
1415    CONTINUE
       IF(SHIFT.GT.0.) CHANGE=CHANGE+1
       GOTO 1250
     ENDIF
*ENDIF(10.5)
*******************************************************************************
*    STEP 3:  MEASURING THE IMPACT OF THE TIME SHIFT                          *
*******************************************************************************
*    GENERAL DESCRIPTION OF STEP 3:
*    BY IMPLEMENTING THE TIME SHIFT, WE HAVE FORCED A TRANSSHIPMENT ARC
*    TO BECOME INFEASIBLE.  THIS MEANS THAT EVERY COMMODITY WHICH FLOWED
*    OVER THAT ARC MUST BE REFLOWED.  IN ORDER FOR A COMMODITY TO BE
*    REFLOWED, THREE CONDITIONS MUST BE SATISFIED:
*      CONDITION 1:  AN ALTERNATE PATH MUST EXIST OVER WHICH THE CARGO
*                    CAN BE FLOWED.
```

```
*     CONDITION 2:  USING THIS ALTERNATE PATH MAINTAINS OR REDUCES THE
*           OVERALL COST OF THE NETWORK.
*     CONDITION 3:  THERE MUST EXIST ENOUGH REMAINING CAPACITY ON THE
*           ALTERNATE PATH TO HANDLE THE REFLOWED COMMODITY.
*   IF ANY ONE OF THESE CONDITIONS IS NOT SATISFIED, THEN THE COMMODITY
*   CANNOT BE REFLOWED AND THE TIME SHIFT MUST BE REVERSED.
*   WE BEGIN OUR CHECK OF THE IMPACT BY FINDING ALL THE PATHS USED IN THE
*   FLOW THAT CONTAIN THE MOST CRITICAL ARC.  FOR EACH PATH WE WILL CHECK
*   THE ABOVE CONDITIONS.  THIS CHECK IS PERFORMED IN SUBROUTINE STEP3.
    DO 1425 Q=1,TRNCNT
*   FIND ALL PATHS THAT CONTAIN THE CRITICAL ARC BEGARC(Q)-ENDARC(Q).
*   CHNGIT IS A 0-1 VARIABLE USED TO DETERMINE WHETHER A CONDITION HAS
*   BEEN VIOLATED.  0=NO VIOLATION, 1=VIOLATION.
    CHNGIT=0
1420  DO 1430 M=1,PATHNUM
       DO 1440 P=2,FLONUM(M)-1
        IF(PATH(M,P-1).EQ.BEGARC(Q).AND.PATH(M,P).EQ.ENDARC(Q))
     C      THEN
*IF(12)
*   A PATH HAS BEEN FOUND WHICH CONTAINS THE CRITICAL ARC.  THE NUMBER
*   OF THE PATH (ITS PLACE IN THE SEQUENCE) IS STORED IN NUMPATH FOR USE
*   IN SUBROUTINE STEP3.
          NUMPATH=M
*   SUBROUTINE STEP3 IS CALLED.
          CALL STEP3(CHNGIT,K,CALLS,TOTAL,Q)
          IF(CHNGIT.EQ.0) THEN
*IF(13)
*   ALL CONDITIONS FOR THIS PATH HAVE BEEN SATISFIED.  PROCEED WITH THE
*   NEXT PATH.
            GOTO 1430
          ELSE
*ELSE(13)
*   A CONDITION FOR THIS PATH HAS NOT BEEN SATISFIED.  PROCEED TO STEP 4.
            GOTO 1450
          ENDIF
*ENDIF(13)
        ENDIF
*ENDIF(12)
1440   CONTINUE
1430  CONTINUE
1425  CONTINUE
*   ALL THE AFFECTED PATHS CONTAINING BEGARC(Q)-ENDARC(Q) HAVE SATISFIED
*   THE CONDITIONS OF STEP 3.  THE TIME SHIFT OF STEP 2 IS PERMANENT.
*   PROCEED TO STEP 1 WITH A NEW MISSION.
*   COMPUTE THE NEW OVERALL COST OF THE FLOW.
    TOTAL=0.
    DO 1460 M=1,PATHNUM
     TIS=NODES(PATH(M,FLONUM(M)-1),2)-NODES(PATH(M,1),2)
     TOTAL=TOTAL+(TIS*PTHFLO(M))
1460  CONTINUE
    CHANGE=CHANGE+1
    GOTO 1250
```

```
**************************************************************************
*   STEP 4:  REVERSAL OF THE TIME SHIFT                    *
**************************************************************************
*   GENERAL DESCRIPTION OF STEP 4:
*   SINCE THE CARGO FLOWED OVER THE CRITICAL PATH CANNOT BE REFLOWED
*   WITHOUT INCREASING OVERALL COST OR REDUCING OVERALL CARGO, THE TIME
*   SHIFT OF STEP 2 HAS HAD AN DAMAGING IMPACT ON THE FLOW PATTERN.
*   WE MUST RESTORE THE NETWORK TO ITS PRE-SHIFT STATE, DETERMINE A TIME
*   SHIFT WHICH WILL MAINTAIN FEASIBILITY, AND RE-SHIFT THE MISSION.
*   RESET THE NODE TIMES FOR EACH NODE ALONG THE MISSION.
1450  DO 1470 V=BEGMSN(K),ENDMSN(K)
      NODES(V,2)=TNODES(V,2)
1470  CONTINUE
*   RESET THE SHIFT TO EQUAL THE MINIMUM OF THE LEAST COST ARC AND THE
*   BEGINNING OF THE MISSION
1480  IF(COST(1).GE.NODES(BEGMSN(K),2)) THEN
*IF(14)
      SHIFT=NODES(BEGMSN(K),2)
    ELSE
*ELSE(14)
      SHIFT=COST(1)
    ENDIF
*ENDIF(14)
    IF(SHIFT.LT.TIMEPS) THEN
      SHIFT=SHIFT+TIMEPS
    ENDIF
    IF((SHIFT-TIMEPS).LE .TIMEPS) THEN
      SHIFT=0.
    ELSE
      SHIFT=SHIFT-TIMEPS
    ENDIF
*   IMPLEMENT THE TIME SHIFT BY ADJUSTING THE TIMES OF THE NODES ALONG
*   THE MISSION
    DO 1490 V=BEGMSN(K),ENDMSN(K)
      NODES(V,2)=NODES(V,2)-SHIFT
1490  CONTINUE
*   SET FEAS EQUAL TO 1 TO INDICATE THAT A FEASIBLE SHIFT WILL OCCUR.
    FEAS=1
*   RESET THE PATHS TO THEIR PRE-SHIFT STATES
    DO 1500 V=1,PATHNUM
      TRNS(V)=TTRNS(V)
      FLONUM(V)=TFLONM(V)
      DO 1510 W=1,TFLONM(V)
        PATH(V,W)=TPATH(V,W)
1510    CONTINUE
1500  CONTINUE
*   RETURN TO THE PORTION OF STEP 2 THAT UPDATES THE COSTS.
*   NOW THAT THE TIMES ASSOCIATED WITH THE MISSION HAVE BEEN
*   RESET AND CHANGED, THE TIMES IN THE COSCAP MATRIX MUST BE UPDATED.
*   CHANGE COSCAP FOR EVERY ARC THAT TERMINATES ON THE MISSION
2320  DO 2330 W=1,N-NUMSNK
      IF(POINT(W,1).LT.0) GOTO 2330
```

```
      DO 2360 V=POINT(W,1),POINT(W,2)
        IF(INT(NODES(INT(COSCAP(V,1)),4)).EQ.K) THEN
*IF(77)
          IF(NODES(W,2).LE.NODES(INT(COSCAP(V,1)),2)) THEN
*IF(79)
            COSCAP(V,5)=NODES(INT(COSCAP(V,1)),2)-
     C                  NODES(W,2)
            IF(ABS(COSCAP(V,6)-COSCAP(V,7)).LE.EPSILON) THEN
*IF(79.5)
              COSCAP(V,5)=INF
            ENDIF
*ENDIF(79.5)
          ELSE
*ELSE(79)
            COSCAP(V,5)=INF
          ENDIF
*ENDIF(79)
        ENDIF
*ENDIF(77)
2360    CONTINUE
2330  CONTINUE
*     WE MUST ALSO CHANGE COSCAP FOR ALL THE ARCS EMANATING FROM THE MISSION
      DO 2370 W=BEGMSN(K),ENDMSN(K)
        IF (POINT(W,1).LT.0) GOTO 2370
       DO 2380 V=POINT(W,1),POINT(W,2)
        IF(NODES(W,2).LE.NODES(INT(COSCAP(V,1)),2)) THEN
*IF(80)
          COSCAP(V,5)=NODES(INT(COSCAP(V,1)),2)-NODES(W,2)
            IF(ABS(COSCAP(V,6)-COSCAP(V,7)).LE.EPSILON) THEN
*IF(80.6)
              COSCAP(V,5)=INF
            ENDIF
*ENDIF(80.6)
        ELSE
*ELSE(80)
          COSCAP(V,5)=INF
         ENDIF
*ENDIF(80)
2380    CONTINUE
2370  CONTINUE
      GOTO 1375
      ELSE
*ELSE(6)
*     IF THE SHIFT IS INFINITY OR 0.0, THEN GOTO TO STEP 1 WITH THE NEXT
*     MISSION.
       GOTO 1250
      ENDIF
*ENDIF(6)
1250  CONTINUE
*******************************************************************************
*     OUTPUT                                                                  *
*******************************************************************************
```

```
      WRITE(*,*)'Changes = ',CHANGE,', Calls to Dijkstra = ',CALLS
*     AS THE PROGRAM PERFORMED STEPS 1-4, IT KEPT TRACK OF THE NUMBER OF
*     CHANGES IT MADE TO THE NETWORK.
*     ONE OPTION IS TO PERFORM STEPS 1-4 UNTIL NO MORE CHANGES CAN BE MADE.
*     INITIAL TESTING SHOWS THAT THIS TAKES HOURS TO ACCOMPLISH.  AN
*     ALTERNATIVE IS TO PERFORM THE STEPS FOR ONLY ONE PASS THROUGH THE
*     MISSION SET AND THEN RETURNING CONTROL TO THE FLOW SUBROUTINES.
*     THE FOLLOWING IF STATEMENT ALLOWS THE USER TO DETERMINE WHICH OPTION
*     IS PREFERRED.  COMMENTING OUT THE LINE FACILITATES THE SINGLE PASS
*     OPTION.
      IF((MSNPAS.LT.PASSES).AND.(CHANGE.GT.0)) GOTO 1240
      IF((MSNPAS.EQ.1).AND.(CHANGE.EQ.0)) TERCRI=1
      OPEN(UNIT=8,FILE='newsched.dat',STATUS='UNKNOWN',FORM='FORMATTED')
*     THE FOLLOWING LOOP DESIGNATES A UNIQUE FILENAME TO EACH ITERATION'S
*     SCHEDULE.
*     THE FILE 'newsched.dat' IS GENERATED FOR USE IN THE FLOW SUBROUTINES.
      TEMP=ITER
      EXT='.c'
      DO 1520 I=2,0,-1
      RATIO=INT(TEMP/(10**I))
       TEMP=TEMP-RATIO*(10**I)
      POS(I+1)=CHAR(48+RATIO)
1520  CONTINUE
      NS=POS(3)//POS(2)//POS(1)
      FILNAM='sched'//NS//EXT
      PTHNAM='paths'//NS//EXT
      WRITE(8,1530)NUMMSN
1530  FORMAT(I3)
      DO 1540 I=1,NUMMSN
        WRITE(8,1550)NUMNODE(I),AC(I),ACCAP(I),MSNCOD(I)
1550    FORMAT(I3,1X,A4,F5.1,2X,A1 )
        DO 1560 J=BEGMSN(I),ENDMSN(I)
          WRITE(8,1570)NODIKO(J,1),NODES(J,2)
1570      FORMAT(A4,1x,F20.10)
1560    CONTINUE
1540  CONTINUE
      OPEN(UNIT=14,FILE=PTHNAM,STATUS='UNKNOWN',FORM='FORMATTED')
*     THIS SECTION WRITES OUT THE PATH A COMMODITY TAKES.  FOR EACH
*     ITERATION THIS PATH SET IS GENERATED.  IF THE FLOW SUBROUTINES
*     ARE UNABLE TO IMPROVE UPON IT, THE MOST RECENT VERSION OF
*     'paths###.c' WILL CONTAIN THE BEST FLOW PATTERN.
      OPEN(UNIT=15,FILE='paths.out',STATUS='OLD',FORM='FORMATTED')
      MODTOT=0.
      MODFLO=0.
       DO 1581 I=1,60
         TRNDIS(I)=0
         TISDIS(I)=0.
1581  CONTINUE
      REFLOW=0.
      DO 1580 I=1,PATHNUM
*     THIS SECTION DETERMINES HOW MUCH CARGO WAS REFLOWED BY COMPARING THE
*     FINAL PATH MATRIX WITH THE INITIAL PATH MATRIX (STORED IN 'paths.out').
```

```
*   IF THE NUMBER OF NODES ON THE PATH HAS CHANGED, THERE HAS OBVIOUSLY
*   BEEN A REFLOW.  IF THE NUMBER OF NODES IS THE SAME, WE MUST CHECK TO
*   SEE IF THE PATH ITSELF HAS CHANGED.  IF IT HAS, THERE HAS BEEN A REFLOW.
      READ(15,*)TISYS,TRANS,FLO,FLNUM,(INPATH(J),J=1,FLNUM)
      IF(FLNUM.NE.FLONUM(I)) THEN
        REFLOW=REFLOW+PTHFLO(I)
      ELSE
        DO 1582 K=1,FLONUM(I)
          IF(INPATH(K).NE.PATH(I,K)) THEN
            REFLOW=REFLOW+PTHFLO(I)
            GOTO 1583
          ENDIF
1582      CONTINUE
      ENDIF
1583    TRNDIS(TRNS(I)+1)=TRNDIS(TRNS(I)+1)+1
      TISPTH=NODES(PATH(I,FLONUM(I)-1),2)-NODES(PATH(I,1),2)
      IF(TISPTH.LT.0.) THEN
        WRITE(*,*)'PATH ',I,' HAS NEGATIVE TIS!'
      ENDIF
      MODTOT=MODTOT+TISPTH*PTHFLO(I)
      MODFLO=MODFLO+PTHFLO(I)
      TISDIS(INT(TISPTH+1))=TISDIS(INT(TISPTH+1))+PTHFLO(I)
      WRITE(14,*)' TIS: ',TISPTH,' FLOW: ',PTHFLO(I)
      WRITE(14,*)' COST OF THIS FLOW: ',TISPTH*PTHFLO(I)
      WRITE(14,*)' NO. OF TRANSSHIPMENTS: ',TRNS(I)
      WRITE(14,*)' ICAO   MSN NO.  TIME   NODE NUMBER'
      WRITE(14,*)' ====   =======  =====  ==========='
      DO 1590 J=1,FLONUM(I)
       WRITE(14,1600)NODIKO(PATH(I,J),1),INT(NODES(PATH(I,J),4)),
     C      NODES(PATH(I,J),2),PATH(I,J)
1600    FORMAT(2X,A4,4X,I4,5X,F5.2,5X,I6)
1590    CONTINUE
      WRITE(14,*)
      WRITE(14,*)
1580  CONTINUE
      MODFLO=0.
      DO 1595 J=NUMCAR+1,N-NUMSNK
        IF(POINT(J,1).LT.0) GOTO 1595
        DO 1596 K=POINT(J,1),POINT(J,2)
          IF(INT(COSCAP(K,1)).GT.(N-NUMSNK)) THEN
            MODFLO=MODFLO+COSCAP(K,7)
          ENDIF
1596      CONTINUE
1595  CONTINUE
      WRITE(14,*)'TOTAL CARGO FLOWED (TONS): ',MODFLO
      WRITE(14,*)'TOTAL COST OF THIS FLOW: ',MODTOT
      WRITE(14,*)'TOTAL CARGO REFLOWED (TONS): ',REFLOW
      WRITE(7,*)'TOTAL CARGO REFLOWED (TONS): ',REFLOW
      WRITE(*,*)'REFLOW THIS ITERATION (tons): ',REFLOW
      WRITE(14,*)
      WRITE(14,*)' TRANSSHIPMENT DISTRIBUTION'
      WRITE(14,*)
```

```
      WRITE(14,*)' NUMBER   OCCURENCES'
      WRITE(14,*)' ----------------'
      WRITE(7,*)
      WRITE(7,*)' TRANSSHIPMENT DISTRIBUTION'
      WRITE(7,*)
      WRITE(7,*)' NUMBER   OCCURENCES'
      WRITE(7,*)' ----------------'
      DO 1503 I=1,60
       IF (TRNDIS(I).GT.0) THEN
        WRITE(14,1505)(I-1),TRNDIS(I)
        WRITE(7,1505)(I-1),TRNDIS(I)
1505    FORMAT(I6,I13)
       ENDIF
1503   CONTINUE
      WRITE(14,*)
      WRITE(14,*)' TIME-IN-SYSTEM DISTRIBUTION'
      WRITE(14,*)
      WRITE(14,*)'  DAYS   TONNAGE'
      WRITE(14,*)' ------------'
      WRITE(7,*)
      WRITE(7,*)' TIME-IN-SYSTEM DISTRIBUTION'
      WRITE(7,*)
      WRITE(7,*)'  DAYS   TONNAGE'
      WRITE(7,*)' ------------'
      DO 1504 I=1,60
       IF (TISDIS(I).GT.0.) THEN
        WRITE(14,1506)(I-1),I,TISDIS(I)
        WRITE(7,1506)(I-1),I,TISDIS(I)
1506    FORMAT(I2,'-',I2,F10.2)
       ENDIF
1504   CONTINUE
      WRITE(5,*)
     CLOSE(8)
     CLOSE(14)
     CLOSE(15)
     RETURN
     END

     SUBROUTINE STEP3(CHNGIT,K,CALLS,TOTAL,Q)

*    THIS SUBROUTINE IS A COMPANION TO SUBROUTINE MODMSN.  IT CHECKS
*    TO SEE IF THE CONDITIONS OF STEP 3 ARE SATISFIED.
*    RECALL THAT THOSE CONDITIONS ARE:
*      CONDITION 1:  AN ALTERNATE PATH MUST EXIST OVER WHICH THE CARGO
*                    CAN BE FLOWED.
*      CONDITION 2:  USING THIS ALTERNATE PATH MAINTAINS OR REDUCES THE
*                    OVERALL COST OF THE NETWORK.
*      CONDITION 3:  THERE MUST EXIST ENOUGH REMAINING CAPACITY ON THE
*                    ALTERNATE PATH TO HANDLE THE REFLOWED COMMODITY.

     INTEGER N,PATHNUM,PATH(500,75),NUMCAR
     INTEGER NUMSNK,NUMNODE(4999),BEGMSN(1000),ENDMSN(1000)
```

```
      INTEGER FLONUM(4999),NUM(4999),NUMMSN,S,T
      INTEGER ITER,POINT(4999,2),ARCNUM,STPATH
      INTEGER FLNUM,FLPATH(4999),NUMPATH
      INTEGER CALLS,CHNGIT,CNT,NUMTRN,LOTRN
      INTEGER K,TRNDIS(60),TRNS(500)
      INTEGER BEGARC(500),ENDARC(500)
      INTEGER CMPCNT,PREV

      REAL COSCAP(89999,8),INF,PRED(4999),DIST(4999)
      REAL NODES(4999,4),TNODES(4999,4),ACCAP(1000)
      REAL PTHFLO(500),TOTAL,TCOSFL,TISDIS(60)
      REAL TIS,TIA,DELTA
      REAL EPSILON,TIMEPS

      CHARACTER NODIKO(4999,2)*4
      CHARACTER AC(1000)*4,MSNCOD(1000)*1

      COMMON /CONSTS/ N, INF, NUMCAR, NUMSNK, NUMMSN, ITER,EPSILON
      COMMON /PARAMS/ MAXIT,MAXALT,MAXTRN,CARCRI,PASSES,SORCRI,TIMEPS
      COMMON /ARRS/ COSCAP, POINT, NODES, NODIKO, TNODES
      COMMON /FLOW/ PRED, DIST, ARCNUM, TRNDIS, TRNS, TISDIS
      COMMON /MARK/ NUMNODE, AC, ACCAP, BEGMSN, ENDMSN, NUM, PATH,
     C         FLONUM, PATHNUM, PTHFLO, MSNCOD
      COMMON /CHNG/ NUMPATH, BEGARC, ENDARC
*************************************************************************
*     CONDITION 1:  FINDING ALTERNATE PATHS                             *
*************************************************************************
*     GENERAL DESCRIPTION OF CONDITION 1:
*     IN SUBROUTINE MODMSN A PATH (NUMBERED NUMPATH) WAS FOUND WHICH
*     CONTAINS THE CRITICAL ARC.  IN ORDER TO DETERMINE WHETHER ALTERNATE
*     PATHS EXIST WE WILL CALL SUBROUTINE DIJKSTRA.
*     IF ANY SUCH ALTERNATE PATHS EXIST, SUBROUTINE DIJKSTRA WILL GENERATE
*     A PARTIAL LIST OF THEM.  THIS PATH REPRESENTS ALL THE CANDIDATES FOR
*     REPLACEMENT PATHS.  IT IS THESE PATHS TO WHICH CONDITIONS 2 AND 3 WILL
*     BE APPLIED.
*     DEFINE S AND T TO BE USED IN DIJKSTRA'S ALGORITHM
      S=PATH(NUMPATH,1)
      T=PATH(NUMPATH,FLONUM(NUMPATH))
*     FORCE THE MOST CRITICAL ARC TO BE INFEASIBLE.  THIS
*     SHOULD HAVE BEEN DONE WITH TIME SHIFT, BUT JUST IN CASE...
      DO 1620 V=POINT(BEGARC(Q),1),POINT(BEGARC(Q),2)
        IF(INT(COSCAP(V,1)).EQ.ENDARC(Q)) THEN
*IF(1)
          COSCAP(V,5)=INF
*       SETTING COSCAP(V,8) TO 0 REMOVES THE ARC FROM FUTURE CONSIDERATION.
          COSCAP(V,8)=0.
        ENDIF
*ENDIF(1)
1620  CONTINUE
*     FILE 'alt.out' CONTAINS THE CANDIDATE ALTERNATE PATHS FROM S TO T.
      OPEN(UNIT=11,FILE='alt2.out',STATUS='UNKNOWN',FORM='FORMATTED')
      IF(S.EQ.0) WRITE(*,*)'S = 0 BEFORE DIJKSTRA'
```

```
      CALL DIJKSTRA(S,T,STPATH,LOTRN)
      CLOSE(11)
      OPEN(UNIT=13,FILE='alt2.out',STATUS='OLD',FORM='FORMATTED')
      CALLS=CALLS+1
      IF(STPATH.EQ.0) THEN
*IF(2)
*     IF NO ALTERNATE PATH CAN BE FOUND, CONDITION 1 HAS NOT BEEN
*     SATISFIED.
*     SET THE VARIABLE CHNGIT TO 1, INDICATING THAT A CONDITION HAS NOT
*     BEEN SATISFIED.  RETURN TO SUBROUTINE MODMSN.
        CHNGIT=1
        RETURN
      ENDIF
*ENDIF(2)
*************************************************************************
*     CONDITION 2:  DOES AN ALTERNATE PATH EXIST WHICH REDUCES OR      *
*              OR MAINTAINS THE OVERALL COST OF THE FLOW?              *
*************************************************************************
*     GENERAL DESCRIPTION OF CONDITION 2:
*     WE NOW HAVE A LIST OF CANDIDATE ALTERNATE PATHS.  WE MUST DETERMINE
*     IF REPLACING THE CURRENT PATH WITH AN ALTERNATE PATH MAINTAINS OR
*     REDUCES THE OVERALL COST OF THE FLOW.  IF NO SUCH PATH EXISTS, THEN
*     CONDITION 2 HAS NOT BEEN SATISFIED AND THE PROGRAM RETURNS TO
*     SUBROUTINE MODMSN.
1640  READ(13,*,END=1650)TIS,TIA,DELTA,NUMTRN,FLNUM,
     c            (FLPATH(V),V=1,FLNUM)
*     CHECK TO SEE IF THE COST OF FLOWING THE CARGO ALONG THIS PATH IS
*     LESS THAN BEFORE THE SHIFT.
*     TCOSFL IS THE OVERALL COST OF THE FLOW GIVEN THAT THE CURRENT PATH
*     HAS BEEN REPLACED BY AN ALTERNATE PATH.
      TCOSFL=0.
      DO 1660 I=1,PATHNUM
        IF(I.NE.NUMPATH) THEN
*IF(3)
          TIS=NODES(PATH(I,FLONUM(I)-1),2)-NODES(PATH(I,1),2)
            TCOSFL=TCOSFL+TIS*PTHFLO(I)
        ENDIF
*ENDIF(3)
1660  CONTINUE
*     ADD COST OF NEW ALTERNATE PATH
      TIS=NODES(FLPATH(FLNUM-1),2)-NODES(FLPATH(1),2)
      TCOSFL=TCOSFL+(PTHFLO(NUMPATH)*TIS)
      IF(TCOSFL.GT.TOTAL) THEN
*IF(4)
*     SINCE USING THIS PATH INCREASES OVERALL COST, WE MUST CONTINUE TO
*     LOOK FOR ANOTHER ALTERNATE PATH.
        GOTO 1640
      ENDIF
*ENDIF(4)
*     IF WE FIND AN ALTERNATE PATH THAT DOES NOT INCREASE THE OVERALL COST,
*     WE PROCEED ON TO CONDITION 3.
*************************************************************************
```

```
*   CONDITION 3:  DOES AN ALTERNATE PATH EXIST WHICH REDUCES OR       *
*             MAINTAINS THE OVERALL COST OF THE FLOW AND              *
*             HAS ENOUGH REMAINING CAPACITY TO HANDLE THE             *
*             CURRENT FLOW?                                           *
*************************************************************************
*   GENERAL DESCRIPTION OF CONDITION 3:
*   WE HAVE NOW FOUND AN ALTERNATE PATH WHICH, IF USED, WILL MAINTAIN OR
*   REDUCE THE OVERALL COST OF THE FLOW.  BUT BEFORE WE CAN PERMANENTLY
*   REPLACE THE CURRENT PATH WITH THIS ALTERNATE, WE MUST VERITY THAT
*   THE ALTERNATE HAS ENOUGH CAPACITY TO HANDLE THE CURRENT FLOW.
*   TO CHECK THE CAPACITIY OF THE ALTERNATE PATH WE NEED ONLY LOOK AT THE
*   CAPACITIES OF THE ARCS WHICH ARE ON THE ALTERNATE PATH AND NOT ON THE
*   ORIGINAL PATH.
    DO 1670 I=2,FLNUM-2
     DO 1675 J=2,FLONUM(NUMPATH)-2
      IF((PATH(NUMPATH,J).EQ.FLPATH(I)).AND.(PATH(NUMPATH,J+1).
   C      EQ.FLPATH(I+1))) THEN
*IF(4.5)
*   A MATCH HAS BEEN FOUND, SO WE MOVE ON TO THE NEXT ARC IN THE
*   ALTERNATE PATH.
        GOTO 1670
      ENDIF
*ENDIF(4.5)
1675   CONTINUE
*   IF THIS SECTION IS REACHED, NO MATCH HAS BEEN FOUND SO WE MUST CHECK
*   THE CAPACITY OF THE APPROPRIATE ARC IN THE ALTERNATE PATH.
    IF(POINT(FLPATH(I),1).EQ.-1) THEN
     WRITE(*,*)'BEFORE 1680, -1'
    ENDIF
     DO 1680 A=POINT(FLPATH(I),1),POINT(FLPATH(I),2)
      IF(INT(COSCAP(A,1)).EQ.FLPATH(I+1)) THEN
*IF(5)
       IF((COSCAP(A,6)-COSCAP(A,7)).LT.PTHFLO(NUMPATH))
   C      THEN
*IF(6)
*   THIS ALTERNATE PATH DOES NOT HAVE ENOUGH CAPACITY.  WE MUST CONTINUE
*   THE SEARCH FOR A PATH WHICH SATISFIES CONDITIONS 2 AND 3.
         GOTO 1640
       ELSE
*ELSE(6)
       ENDIF
*ENDIF(6)
      ENDIF
*ENDIF(5)
1680   CONTINUE
1670  CONTINUE
*   WE HAVE FOUND A PATH THAT SATISFIES CONDITIONS 2 AND 3.  WE MUST
*   NOW REPLACE THE CURRENT PATH WITH THIS NEW ALTERNATE PATH.
*   IF THE NUMBER OF TRANSSHIPMENTS ON THIS NEW ALTERNATE PATH EXCEEDS
*   0, THEN WE WILL CALL SUBROUTINE COMPRESS TO COMPRESS IT TO ITS
*   ACTUAL PATH.
*************************************************************************
```

```
*   CALLING SUBROUTINE COMPRESS                                    *
*********************************************************************************
*   ESTABLISH THE PREDECESSOR ARRAY
    DO 1671 I=2,FLNUM
      PRED(FLPATH(I))=FLPATH(I-1)
1671  CONTINUE
    PRED(S)=-1
    if((ITER.EQ.3).AND.(K.EQ.120)) THEN
      WRITE(*,*)'PATH = ',NUMPATH,' PRED = ',
   c     (PRED(FLPATH(A)),A=1,FLNUM)
    ENDIF
    CALL COMPRESS(S,T,STPATH,NUMTRN)
    IF (NUMTRN.GT.MAXTRN) GOTO 1640
*   WE MUST GLEAN FROM THE NEW PREDECESSOR ARRAY (RETURNED BY COMPRESS)
*   WHAT THE NEW COMPRESSED PATH IS.
    FLNUM=0
    CMPCNT=T
1672  IF(PRED(CMPCNT).EQ.-1) THEN
      FLNUM=FLNUM+1
      GOTO 1673
    ELSE
      FLNUM=FLNUM+1
      CMPCNT=PRED(CMPCNT)
      GOTO 1672
    ENDIF
1673  CONTINUE
    FLPATH(FLNUM)=T
    FLPATH(1)=S
    PREV=T
    DO 1674 I=FLNUM-1,2,-1
      FLPATH(I)=PRED(PREV)
      PREV=FLPATH(I)
1674  CONTINUE
*   THE NEW PATH HAS NOW BEEN COMPRESSED.
*   NOW THAT WE KNOW THE NEW PATH CAN HANDLE THE NEW FLOW, WE MUST
*   REMOVE THE OLD FLOW FROM THE PATH AND REPLACE IT WITH THE NEW.
*********************************************************************************
*   PATH REPLACEMENT                                               *
*********************************************************************************
*   GENERAL DESCRIPTION OF PATH REPLACEMENT:
*   WE REPLACE THE CURRENT PATH WITH THE NEW PATH BY PERFORMING SEVERAL
*   STEPS:
*     STEP 1:  ADD THE CURRENT FLOW BACK TO THE OLD PATH
*     STEP 2:  DETERMINE IF THE ARCS ON THE OLD PATH ARE USED ELSEWHERE.
*              IF NOT, REMOVE THESE ARCS FROM CONSIDERATION IN
*              SUBROUTINE MODMSN STEP 1.
*     STEP 3:  SWITCH THE PATHS.
*     STEP 4:  INCREMENT THE FLOW OF THE NEW PATH APPROPRIATELY.
*     STEP 5:  MARK THE NEW PATH WITH APPROPRIATE MISSION IDENTIFIERS.
*********************************************************************************
*   PATH REPLACEMENT STEP 1:  REMOVE THE CURRENT FLOW FROM THE OLD PATH  *
*********************************************************************************
```

```
      DO 1690 I=1,FLONUM(NUMPATH)-1
        IF(POINT(PATH(NUMPATH,I),1).EQ.-1) THEN
        WRITE(*,*)'BEFORE 1700, -1'
        ENDIF
        DO 1700 J=POINT(PATH(NUMPATH,I),1),POINT(PATH(NUMPATH,I),2)
          IF(INT(COSCAP(J,1)).EQ.PATH(NUMPATH,I+1)) THEN
*IF(8)
            COSCAP(J,7)=COSCAP(J,7)-PTHFLO(NUMPATH)
          ENDIF
*ENDIF(8)
1700    CONTINUE
1690  CONTINUE
*****************************************************************************
*    PATH REPLACEMENT STEP 2:  DETERMINE IF ARCS ARE USED ELSEWHERE      *
*****************************************************************************
      DO 1710 I=1,FLONUM(NUMPATH)-1
*     CNT IS A 0-1 VARIABLE USED TO DETERMINE IF AN ARC IS USED ELSEWHERE.
*     0=NOT USED, 1=USED.
        CNT=0
        DO 1720 J=1,PATHNUM
          IF(J.NE.NUMPATH) THEN
*IF(9)
            DO 1730 V=1,FLONUM(J)-1
              IF(PATH(NUMPATH,I).EQ.PATH(J,V).AND.
     C           PATH(NUMPATH,I+1).EQ.PATH(J,V+1)) THEN
*IF(10)
                CNT=1
              ENDIF
*ENDIF(10)
1730        CONTINUE
          ENDIF
*ENDIF(9)
1720    CONTINUE
        IF(CNT.EQ.0) THEN
          IF(POINT(PATH(NUMPATH,I),1).EQ.-1) THEN
            WRITE(*,*)'BEFORE 1740, -1'
          ENDIF
*IF(11)
          DO 1740 V=POINT(PATH(NUMPATH,I),1),POINT(PATH(NUMPATH,I),2)
            IF(INT(COSCAP(V,1)).EQ.PATH(NUMPATH,I+1)) THEN
*IF(12)
              COSCAP(V,8)=0.
            ENDIF
*ENDIF(12)
1740      CONTINUE
        ENDIF
*ENDIF(11)
1710  CONTINUE
*****************************************************************************
*    PATH REPLACEMENT STEP 3:  SWITCH THE PATHS                       *
*****************************************************************************
      TRNS(NUMPATH)=NUMTRN
```

```
      FLONUM(NUMPATH)=FLNUM
      DO 1750 I=1,FLONUM(NUMPATH)
        PATH(NUMPATH,I)=FLPATH(I)
1750  CONTINUE
*************************************************************************
*    PATH REPLACEMENT STEP 4:  INCREMENT THE FLOW OF THE NEW PATH
*************************************************************************
      DO 1760 I=1,FLONUM(NUMPATH)-1
        IF(POINT(PATH(NUMPATH,I),1).EQ.-1) THEN
          WRITE(*,*)'BEFORE 1770, -1'
        ENDIF
        DO 1770 J=POINT(PATH(NUMPATH,I),1),POINT(PATH(NUMPATH,I),2)
          IF(INT(COSCAP(J,1)).EQ.PATH(NUMPATH,I+1)) THEN
*IF(13)
            COSCAP(J,7)=COSCAP(J,7)+PTHFLO(NUMPATH)
            IF(ABS(COSCAP(J,6)-COSCAP(J,7)).LE.EPSILON) THEN
*IF(13.5)
              COSCAP(J,5)=INF
            ENDIF
*ENDIF(13.5)
          ENDIF
*ENDIF(13)
1770    CONTINUE
1760  CONTINUE
*************************************************************************
*    PATH REPLACEMENT STEP 5:  MARK THE NEW PATH WITH APPROPRIATE MISSION *
*                      IDENTIFIERS                    *
*************************************************************************
      DO 1780 I=1,FLONUM(NUMPATH)-2
        IF(POINT(PATH(NUMPATH,I),1).EQ.-1) THEN
          WRITE(*,*)'BEFORE 1790, -1'
        ENDIF
        DO 1790 J=POINT(PATH(NUMPATH,I),1),POINT(PATH(NUMPATH,I),2)
          IF(INT(COSCAP(J,1)).EQ.PATH(NUMPATH,I+1)) THEN
*IF(14)
            DO 1800 V=1,NUMMSN
              IF(INT(COSCAP(J,1)).GE.BEGMSN(V).AND.
     C           INT(COSCAP(J,1)).LE.ENDMSN(V)) THEN
*IF(15)
                COSCAP(J,8)=V*1.0
              ENDIF
*ENDIF(15)
1800        CONTINUE
          ENDIF
*ENDIF(14)
1790    CONTINUE
1780  CONTINUE
*    WE NOW HAVE REPLACED THE CURRENT PATH WITH THE NEW ALTERNATE PATH.
      DO 1785 V=POINT(PATH(NUMPATH,1),1),POINT(PATH(NUMPATH,1),2)
        IF(INT(COSCAP(V,1)).EQ.PATH(NUMPATH,2)) THEN
        ENDIF
1785  CONTINUE
```

```
*    WE CAN CLOSE 'alt.out' AND RETURN TO SUBROUTINE MODMSN
     CLOSE(13)
     RETURN
*    IF WE HAVE EXHAUSTED THE LIST OF CANDIDATE ALTERNATE PATHS AND FOUND
*    NONE WHICH SATISFY CONDITIONS 2 AND 3, WE SET CHNGIT TO 1 TO INDICATE
*    THAT A FEASIBLE SHIFT WILL OCCUR.  WE CLOSE 'alt.out' AND RETURN TO
*    SUBROUTINE MODMSN.
1650 CHNGIT=1
     CLOSE(13)
     RETURN
     END

     SUBROUTINE COUNTER()

*************************************************************

     INTEGER N,PATHNUM,PATH(500,75),NUMCAR
     INTEGER NUMSNK,NUMNODE(4999),BEGMSN(1000),ENDMSN(1000)
     INTEGER FLONUM(4999),NUM(4999),NUMMSN,K
     INTEGER ITER,POINT(4999,2),ARCNUM,CNT
     INTEGER TRNDIS(60),TRNS(500)

     REAL COSCAP(89999,8),INF,PRED(4999),DIST(4999)
     REAL NODES(4999,4),TNODES(4999,4),ACCAP(1000)
     REAL TIME(1000),TEMTIME,PTHFLO(500)
     REAL EPSILON,TISDIS(60)

     CHARACTER NODIKO(4999,2)*4
     CHARACTER AC(1000)*4,ICAO(1000)*4
     CHARACTER TEMICAO*4,TEMACFT*4,ACFT(1000)*4,MSNCOD(1000)*1

     COMMON /CONSTS/ N, INF, NUMCAR, NUMSNK, NUMMSN, ITER,EPSILON
     COMMON /ARRS/ COSCAP, POINT, NODES, NODIKO, TNODES
     COMMON /FLOW/ PRED, DIST, ARCNUM, TRNDIS, TRNS, TISDIS
     COMMON /MARK/ NUMNODE, AC, ACCAP, BEGMSN, ENDMSN, NUM, PATH,
    C          FLONUM, PATHNUM, PTHFLO, MSNCOD

     OPEN(UNIT=11,FILE='count.out',STATUS='UNKNOWN',FORM='FORMATTED')
*    SORT ON ICAO
     DO 1810 I=1,NUMMSN
       ICAO(I)=NODIKO(BEGMSN(I),1)
       TIME(I)=NODES(BEGMSN(I),2)
       ACFT(I)=AC(I)
1810 CONTINUE
     DO 1820 I=1,NUMMSN
       TEMICAO='ZZZZ'
       DO 1830 J=I,NUMMSN
         IF(ICAO(J).LT.TEMICAO) THEN
           TEMICAO=ICAO(J)
           K=J
         ENDIF
1830     CONTINUE
```

```
*     SWITCH
      TEMICAO=ICAO(I)
      TEMTIME=TIME(I)
      TEMACFT=ACFT(I)
      ICAO(I)=ICAO(K)
      TIME(I)=TIME(K)
      ACFT(I)=ACFT(K)
      ICAO(K)=TEMICAO
      TIME(K)=TEMTIME
      ACFT(K)=TEMACFT
1820  CONTINUE
*   NOW SORT ON ACFT AT EACH ICAO
    DO 1840 I=1,NUMMSN
      TEMACFT='ZZZZ'
      DO 1850 J=I,NUMMSN
        IF(ICAO(J).EQ.ICAO(I)) THEN
          IF(ACFT(J).LT.TEMACFT) THEN
            TEMACFT=ACFT(J)
            K=J
          ENDIF
        ELSE
          GOTO 1850
        ENDIF
1850    CONTINUE
      TEMICAO=ICAO(I)
      TEMTIME=TIME(I)
      TEMACFT=ACFT(I)
      ICAO(I)=ICAO(K)
      TIME(I)=TIME(K)
      ACFT(I)=ACFT(K)
      ICAO(K)=TEMICAO
      TIME(K)=TEMTIME
      ACFT(K)=TEMACFT
1840  CONTINUE
*   NOW SORT ON TIME WITHIN EACH ICAO-ACFT PAIR
    DO 1860 I=1,NUMMSN
      TEMTIME=INF
      DO 1870 J=I,NUMMSN
        IF(ICAO(J).EQ.ICAO(I)) THEN
          IF(ACFT(J).EQ.ACFT(I)) THEN
            IF(TIME(J).LT.TEMTIME) THEN
              TEMTIME=TIME(J)
              K=J
            ENDIF
          ELSE
            GOTO 1870
          ENDIF
        ELSE
            GOTO 1870
        ENDIF
1870    CONTINUE
      TEMICAO=ICAO(I)
```

```
      TEMTIME=TIME(I)
      TEMACFT=ACFT(I)
      ICAO(I)=ICAO(K)
      TIME(I)=TIME(K)
      ACFT(I)=ACFT(K)
      ICAO(K)=TEMICAO
      TIME(K)=TEMTIME
      ACFT(K)=TEMACFT
 1860 CONTINUE
*    NOW COUNT THE NUMBER OF TAKEOFFS AT EACH ICAO FOR EACH ACFT
*    AT EACH TIME
     CNT=1
     DO 1880 I=2,NUMMSN
       IF(ICAO(I).EQ.ICAO(I-1)) THEN
         IF(ACFT(I).EQ.ACFT(I-1)) THEN
           IF(TIME(I).EQ.TIME(I-1)) THEN
             CNT=CNT+1
           ELSE
             WRITE(11,1890)ICAO(I-1),ACFT(I-1),TIME(I-1),CNT
             CNT=1
           ENDIF
         ELSE
           WRITE(11,1890)ICAO(I-1),ACFT(I-1),TIME(I-1),CNT
           CNT=1
         ENDIF
       ELSE
         WRITE(11,1890)ICAO(I-1),ACFT(I-1),TIME(I-1),CNT
         CNT=1
       ENDIF
 1880 CONTINUE
     WRITE(11,1890)ICAO(NUMMSN),ACFT(NUMMSN),TIME(NUMMSN),CNT
 1890 FORMAT(A4,2X,A4,2X,F7.3,2X,I4)
     CLOSE(11)
     RETURN
     END

********** END - MAIN PROGRAM **********
```

Appendix C: Program Creating the Initial Schedule

This appendix is the program "makesked.f", which generates the initial schedule "schedule.dat" using the input files contained in Appendices D-G.

```
PROGRAM MAKESKED

*    CREATED 28 OCTOBER 1993
*    THIS PROGRAM GENERATES THE SCHEDULE, CREATED AS A FILE CALLED
*    'tempsked.dat'.  IT USES INPUT FROM 'schedule.raw' AND 'routes.dat'

     CHARACTER STA(999,20)*4,STATYP(999,20)*1,ICAO(999)*4,AC(999)*4
     CHARACTER BASE1(999)*4,BASE2(999)*4,JUNK5*4,MSNCOD(999)*1
     INTEGER MSNCNT,ROUCNT,NODENUM,ROUTE(999),ROUNUM(999),TOTMSN
     INTEGER ACTYPE(999),LEG
     REAL START(999),GTIME(999),GRND(9),TIME(999),RON(999)
     REAL REMAIN(9),JUNK1,JUNK2,JUNK3,JUNK4,FLY(999),FLYTIME
     REAL SPEED(9),CAP(9),ACCAP(999)

     OPEN(UNIT=1,FILE='jet.dat',STATUS='OLD',FORM='FORMATTED')
     OPEN(UNIT=2,FILE='fly.dat',STATUS='OLD',FORM='FORMATTED')
     OPEN(UNIT=3,FILE='schedule.raw',STATUS='OLD',FORM='FORMATTED')
     OPEN(UNIT=4,FILE='routes.dat',STATUS='OLD',FORM='FORMATTED')
     OPEN(UNIT=5,FILE='schedule.dat',STATUS='UNKNOWN',FORM=
    C    'FORMATTED')

*    THE FOLLOWING LOOP READS THE SPEED FACTORS FOR EACH AIRCRAFT.
*    THE ORDER IS THE SAME AS THE 'TYPES' IN NEXT SECTION.  SPEED IS
*    USED LATER TO CALCULATE THE FLYING TIME ALONG A PARTICULAR LEG
*    FOR A PARTICULAR AIRCRAFT.  THE FLYING TIMES, WHICH WILL LATER
*    BE READ FROM 'fly.dat' ARE FOR THE C-141 ONLY.  SPEED ACTS AS THE
*    CONVERSION FACTOR FOR ALL OTHER AIRCRAFT.  SPEED*(C141
*    FLYTIME)=FLYING TIME FOR THE AIRCRAFT.

     READ(1,*)
     DO 10 I=1,9
       READ(1,*)JUNK1,JUNK5,SPEED(I),CAP(I)
10   CONTINUE

*    THE FOLLOWING CODES WILL BE USED IN THE NEXT SECTION TO
*    ESTABLISH AN APPROPRIATE GROUND TIME AND RON TIME FOR EACH
*    MISSION:
*    C005 = TYPE 1
*    C141 = TYPE 2
*    C130 = TYPE 3
```

```
*    DC08 = TYPE 4
*    DC10 = TYPE 5
*    B747 = TYPE 6
*    KC10 = TYPE 7
*    C017 = TYPE 8
*    KC135= TYPE 9

*    THE FOLLOWING STANDARD GROUND TIMES AND REMAIN OVER NIGHT
*    (RON) TIMES ARE READ IN FROM 'jet.dat'. TO CONVERT THE TIME TO DAYS,
*    THE TIMES IN 'jet.dat' WILL BE DIVIDED BY 24 IN A LATER SECTION.

     DO 20 I=1,9
       READ(1,*)JUNK1,GRND(I),JUNK2,JUNK3,JUNK4,REMAIN(I)
20   CONTINUE

*    THE FOLLOWING LOOP READS IN THE FLYING TIME DATA FROM 'fly.dat'.
*    THE FIRST TWO COLUMNS ARE THE BASES OF THE LEG.  THE NEXT
*    COLUMN IS THE FLYING TIME FOR A C141.

     LEG=0
     DO 30 I=1,1000000
       READ(2,*,END=40)BASE1(I),BASE2(I),JUNK1,FLY(I)
       LEG=LEG+1
30   CONTINUE
40   CONTINUE

*    THE FOLLOWING LOOP READS IN THE DATA FROM 'routes.dat'.  THE FIRST
*    COLUMN OF 'routes.dat' IS THE ROUTE NUMBER.  THE FOLLOWING
*    COLUMNS CONTAIN THE BASES ON THE ROUTE ALONG WITH THE TYPE OF
*    BASE, WHERE
*    1 = MISSION ORIGIN
*    4 = STANDARD GROUND TIME
*    6 = RON
*    9 = MISSION TERMINATION

     ROUCNT=0
     DO 50 I=1,100000
       READ(4,60,END=70)ROUNUM(I),(STA(I,J),STATYP(I,J),J=1,20)
60      FORMAT(I3,1X,20(1X,A4,A1))
       ROUCNT=ROUCNT+1
50   CONTINUE
70   CONTINUE

*    THE FOLLOWING LOOP READS IN THE MISSION DATA FROM 'schedule.raw'.
*    THE FIRST COLUMN IS THE ROUTE NUMBER, THE SECOND COLUMN IS THE
*    AIRCRAFT TYPE, AND THE THIRD COLUMN IS THE START TIME OF THE
*    MISSION.
```

```
      MSNCNT=0
      TOTMSN=0
      DO 80 I=1,100000
        MSNCOD(I)='C'

*     THE FOLLOWING READ SHOULD BE USED IF 'schedule.raw' IS EVER UPDATED
*     TO INCLUDE A CODE TO DISTINGUISH CARGO REQUIREMENTS 'C' AND
*     FREQUENCY REQUIREMENTS 'F'.  AS IS, THE PROGRAM ASSUMES ALL ARE
*     'C' UNLESS OTHERWISE INSTRUCTED.

c       READ(3,*,END=90)ROUTE(I),AC(I),START(I),MSNCOD(I)
        READ(3,*,END=90)ROUTE(I),AC(I),START(I)
        IF(AC(I).EQ.'C005') ACTYPE(I)=1
        IF(AC(I).EQ.'C141') ACTYPE(I)=2
        IF(AC(I).EQ.'C130') ACTYPE(I)=3
        IF(AC(I).EQ.'DC08') ACTYPE(I)=4
        IF(AC(I).EQ.'DC10') ACTYPE(I)=5
        IF(AC(I).EQ.'B747') ACTYPE(I)=6
        IF(AC(I).EQ.'KC10') ACTYPE(I)=7
        IF(AC(I).EQ.'C017') ACTYPE(I)=8
        IF(AC(I).EQ.'K135') ACTYPE(I)=9
        TOTMSN=TOTMSN+1
        GTIME(I)=GRND(ACTYPE(I))/24.0
        RON(I)=REMAIN(ACTYPE(I))/24.0
        ACCAP(I)=CAP(ACTYPE(I))
        IF(MSNCOD(I).NE.'F') MSNCOD(I)='C'

*     THE FOLLOWING LOOP COUNTS THE NUMBER OF MISSIONS.  THIS VALUE
*     WILL BE THE TOP LINE OF THE OUTPUT.

        DO 100 J=1,ROUCNT
          IF(ROUTE(I).EQ.ROUNUM(J)) THEN
            MSNCNT=MSNCNT+1
            GOTO 80
          ENDIF
100     CONTINUE
80    CONTINUE
90    CONTINUE

*     THE FOLLOWING LOOPS COMBINE THE ABOVE INFORMATION INTO
*     'schedule.dat'.

      WRITE(5,*)MSNCNT
      DO 110 I=1,TOTMSN
        NODENUM=0
        DO 120 J=1,ROUCNT
```

```
      IF(ROUTE(I).EQ.ROUNUM(J)) THEN
        DO 130 K=1,20
          NODENUM=NODENUM+1
          ICAO(NODENUM)=STA(J,K)
          IF(STATYP(J,K).NE.' '.AND.STATYP(J,K).NE.'1') THEN

*   THE FLYING TIME IS DEFAULTED TO .3 DAYS.  SOME LEGS DO NOT APPEAR
*   IN FLY.DAT, SO A DEFAULT VALUE MUST BE USED.

            FLYTIME=.3
            DO 140 M=1,LEG
              IF(((ICAO(NODENUM).EQ.BASE2(M)).AND.
     C              (ICAO(NODENUM-1).EQ.BASE1(M)))) THEN
                FLYTIME=(FLY(M)*SPEED(ACTYPE(I)))/24.0
              ENDIF
140           CONTINUE
            IF(FLYTIME.EQ..3) THEN
              DO 150 M=1,LEG
                IF(((ICAO(NODENUM).EQ.BASE1(M)).AND.
     C                (ICAO(NODENUM-1).EQ.BASE2(M)))) THEN
                  FLYTIME=(FLY(M)*SPEED(ACTYPE(I)))/24.0
                ENDIF
150             CONTINUE
            ENDIF
c           IF(FLYTIME.EQ..3) THEN
c             WRITE(*,*)'CHECK ROUTE ',ROUTE(I)
c           ENDIF
          ENDIF
          IF(STATYP(J,K).EQ.'1') THEN
            TIME(NODENUM)=START(I)
          ELSE
          IF(STATYP(J,K).EQ.'9') THEN
            TIME(NODENUM)=TIME(NODENUM-1)+FLYTIME

*   THIS SECTION WRITES TO THE OUTPUT FILE.

            WRITE(5,155)NODENUM,AC(I),ACCAP(I),MSNCOD(I)
155           FORMAT(I3,1X,A4,F5.1,2X,A1)
            DO 160 L=1,NODENUM
              WRITE(5,*)ICAO(L),TIME(L)
160           CONTINUE
          ELSE
          IF(STATYP(J,K).EQ.'4') THEN
            TIME(NODENUM)=TIME(NODENUM-1)+FLYTIME
            NODENUM=NODENUM+1
            TIME(NODENUM)=TIME(NODENUM-1)+GTIME(I)
            ICAO(NODENUM)=STA(J,K)
```

```
            ELSE
            IF(STATYP(J,K).EQ.'6') THEN
              TIME(NODENUM)=TIME(NODENUM-1)+FLYTIME
              NODENUM=NODENUM+1
              TIME(NODENUM)=TIME(NODENUM-1)+RON(I)
              ICAO(NODENUM)=STA(J,K)
            ENDIF
            ENDIF
            ENDIF
            ENDIF
130         CONTINUE
          ENDIF
120       CONTINUE
110   CONTINUE
      WRITE(*,*)'PROGRAM COMPLETED'
      END
```

Appendix D: STORM/CARGPREP Schedule for the Sub-Problem

This appendix is an extract of the output file, "schedule.raw" generated by the STORM and CARGPREP models. The first column is the number of the route to be flown by the aircraft in column 2. The final column is the start time of the mission (days). For example, line 1 indicates that a C005 is to fly a mission along route 19 beginning at the 0.1 day point in the planning horizon.

```
 19  C005   0.1
 19  C005  15.1
 23  C005   1.2
 37  C005   2.3
 56  C005   3.4
 58  C005   4.5
 58  C005  12.0
 58  C005  19.5
 58  C005  27.0
       .
       .
       .
252  KC10   3.3
252  KC10   5.6
252  KC10   7.9
252  KC10  10.2
252  KC10  12.5
252  KC10  14.8
252  KC10  17.1
252  KC10  19.5
252  KC10  21.8
252  KC10  24.1
252  KC10  26.4
252  KC10  28.7
252  KC10   1.0
253  KC10   4.4
```

Appendix E: Aircraft Capacities and Ground/RON Times

This appendix shows the contents of "jet.dat", containing aircraft information which was used in the program "makesked.f" (Appendix C). For lines 2 - 9, the columns of interest are columns 3 and 4. Column 3 is the speed conversion factor. Because all flight times in file "fly.dat" (Appendix F) are given in terms of the C141, the speed conversion factor was needed to adjust flying times for the other aircraft. Column 4 is the capacity (tons) of each aircraft, based on AMC/XPYR's use of 1.5 tons per pallet instead of 2.3 tons as stated in AFR 76-1.

For lines 10 - 17, column 5 is the authorized ground time (hours) and Column 6 is the Remain-Over-Night (RON) time (hours) for the aircraft. The aircraft follow the order in lines 2 - 9. For example, line 10 corresponds to a C005, line 11 to a C141, etc.

8	175						
20	C005	0.97	54.00				
30	C141	1.00	20.00				
20	C130	1.39	9.00				
10	DC08	0.93	27.00				
15	DC10	0.92	45.00				
10	B747	0.91	63.00				
10	KC10	0.92	33.00				
60	C017	0.97	28.00				
0.00	4.25	4.25	4.25	4.25	18.25	4.25	4.25
0.00	3.25	3.25	3.25	3.25	17.25	3.25	3.25
0.00	2.25	2.25	2.25	2.25	16.25	2.25	2.25
0.00	3.00	3.00	3.00	3.00	16.00	3.00	3.00
0.00	4.00	4.00	4.00	4.00	16.00	4.00	4.00
0.00	4.00	4.00	4.00	4.00	16.00	4.00	4.00
0.00	3.25	3.25	3.25	3.25	17.25	3.25	3.25
0.00	3.25	3.25	3.25	3.25	17.25	3.25	3.25

Appendix F: Flying Times Between Airbases

This appendix is an extract of the file "fly.dat", which contains the times required for a C141 to fly from the airbase in column 1 to the airbase in column 2. This data was used in "makesked.f" to generate the times associated with each airbase in the initial schedule. Note that flight times are not necessarily commutative. For example, a flight from KDOV (Dover AFB) to EDAR (Ramstein AB) requires 8.2 hours, while the flight from EDAR to KDOV is 9.5 hours.

```
KDOV KCHS  1.5 1.5 1.5 1.5 1.5 1.5 1.5
KDOV KTIK  3.3 3.3 3.3 3.3 3.3 3.3 3.3
KDOV KWRI  0.7 0.7 0.7 0.7 0.7 0.7 0.7
KDOV EDAR  8.2 8.2 8.2 8.2 8.2 8.2 8.2
KDOV EDAF  7.9 7.9 7.9 7.9 7.9 7.9 7.9
KDOV EDAF  8.2 8.2 8.2 8.2 8.2 8.2 8.2
KDOV EGUN  7.1 7.1 7.1 7.1 7.1 7.1 7.1
KDOV LETO  7.8 7.8 7.8 7.8 7.8 7.8 7.8
      .
      .
      .
EDAR KDOV  9.5 9.5 9.5 9.5 9.5 9.5 9.5
EDAR KWRI  9.6 9.6 9.6 9.6 9.6 9.6 9.6
EDAR EGUN  1.5 1.5 1.5 1.5 1.5 1.5 1.5
EDAR LETO  2.6 2.6 2.6 2.6 2.6 2.6 2.6
EDAR LPLA  4.6 4.6 4.6 4.6 4.6 4.6 4.6
EDAR LIPA  1.5 1.5 1.5 1.5 1.5 1.5 1.5
EDAR LIRN  2.1 2.1 2.1 2.1 2.1 2.1 2.1
EDAR LIRP  2.9 2.9 2.9 2.9 2.9 2.9 2.9
EDAR LICZ  2.6 2.6 2.6 2.6 2.6 2.6 2.6
EDAR LTAG  4.4 4.4 4.4 4.4 4.4 4.4 4.4
EDAR LLBG  4.3 4.3 4.3 4.3 4.3 4.3 4.3
EDAR HECA  4.7 4.7 4.7 4.7 4.7 4.7 4.7
      .
      .
      .
RPMB FJDG  7.6 7.6 7.6 7.6 7.6 7.6 7.6
RPMB RPMK  0.5 0.5 0.5 0.5 0.5 0.5 0.5
RPMB RJTY  4.1 4.1 4.1 4.1 4.1 4.1 4.1
RPMB PGUA  3.8 3.8 3.8 3.8 3.8 3.8 3.8
RPMB RODN  2.4 2.4 2.4 2.4 2.4 2.4 2.4
VTBD FJDG  5.3 5.3 5.3 5.3 5.3 5.3 5.3
WSAP FJDG  5.0 5.0 5.0 5.0 5.0 5.0 5.0
WSAP RPMK  3.6 3.6 3.6 3.6 3.6 3.6 3.6
```

Appendix G: Routes in the E/SWA Sub-Problem

This appendix shows the file "route.dat", which contains a list of the routes used as input to "makesked.f". The first column is the route number. The remaining columns are the ICAO designations of the airbases along the route. Appended to each ICAO is a numeric suffix which identifies the airbase type. Suffix 1 indicates the origin airbase for the mission, suffix 4 indicates that the aircraft will spend its authorized ground time at the airbase, suffix 6 indicates that the aircraft will remain at the airbase for its authorized RON time, and suffix 9 indicates the final destination of the mission.

```
  3   EXXX1 KTIK4 CYQX4 EDAR4 EXXX9
 56   KSUU1 KTIK4 KDOV6 EDAF6 KDOV6 KTIK4 KSUU9
 58   KSUU1 KTIK4 KDOV6 EDAR6 KDOV6 KTIK4 KSUU9
 59   KSUU1 KTIK4 KDOV6 EGUN6 EDAR4 EDAF6 KCHS6 KTIK4 KSUU9
137   KXXX1 KTIK4 EDAF4 KDOV4 KTIK4 KXXX9
180   KDOV1 EDAF6 KDOV9
181   KDOV1 EDAR6 KDOV9
196   KCHS1 KNGU4 LPLA6 GOOY6 GLRB4 FZAA6 FTTJ4 FZAA6 GOOY4 LPLA6
      KNGU4 KCHS9
200   KDOV1 EDAR6 OJAF6 EDAR6 KDOV9
202   KCHS1 KNGU4 BIKF6 EGUN4 KCHS9
203   KDOV1 KCHS4 KNGU4 BIKF6 EGUN4 KDOV9
216   KCHS1 KNGU4 LERT6 LICZ4 OBBI4 OMFJ6 OBBI4 LICZ6 LERT4 LPLA6
      KNGU4 KCHS9
224   KDOV1 EDAF6 OEDR4 EDAF6 KDOV9
225   KSUU1 KTIK4 KWRI6 LPLA4 EDAF6 KWRI6 KTIK4 KSUU9
230   EDAF1 LETO4 LIPA6 EDAR4 EGUN4 EDAF9
231   EDAF1 EGUN4 EDAR6 LIPA4 LETO4 EDAF9
235   EDAF1 OKBK4 OEDR6 OERY4 EDAF9
237   EDAF1 LTAG4 EDAF9
239   EDAR1 LTAG4 EDAR9
241   KDOV1 LETO6 KDOV9
242   KWRI1 LPLA6 KWRI9
249   EGUN1 EDAR4 LIRP4 LIPA6 LETO4 EDAR4 EGUN9
251   EGUN1 EDAF4 LIPA6 LGIR4 LCRA4 LTAG6 LCRA4 LGIR4 LIPA6 EDAF4
      EGUN9
252   KDOV1 EDAR4 LTAG4 EDAR4 KDOV9
255   KDOV1 KNGU4 LERT6 OBBI4 LICZ6 LERT6 KNGU4 KDOV9
259   KCHS1 KNGU4 LERT6 LIRN4 LICZ6 LIRN4 LERT6 KNGU4 KCHS9
260   KCHS1 KNGU4 LERT6 LIRN4 LERT6 KNGU4 KCHS9
262   EDAF1 EGUN4 EDAR4 LIPA4 LETO4 EDAF4 LTAG6 EDAF4 LETO4 LIPA4
      EDAR4 EGUN4 EDAF9
264   EDAF1 LIRN4 LICZ4 LERT6 LICZ4 LIRN4 EDAF9
265   KCHS1 KNGU4 LERT6 LIRN4 LICZ4 OBBI6 OMFJ4 OBBI4 LICZ6 LIRN4
      LERT6 LPLA4 KNGU4 KCHS9
```

266 EDAF1 LIRN4 LICZ4 LIRN4 EDAF9
269 KDOV1 EDAF4 OERY6 EDAF4 KDOV9
270 KWRI1 LPLA4 EDAR6 LPLA4 KWRI9
271 EDAF1 OEDR6 EDAF9
292 EDAF1 EDAR4 EDAF9
293 KDOV1 EDAR4 LLBG4 EDAR4 KDOV9
294 KNGU1 LETO4 LICZ4 HSSS4 HKNA4 LICZ4 LPLA4 KNGU9

Appendix H: Commodities of the E/SWA Sub-Problem

This appendix is the file "cargo.dat", which contains the cumulative amounts of the commodities which arrive during the first week of the planning horizon. The first two columns are the ICAO designations of the OD pair. The third column indicates the first week of the planning horizon (1 equals week 2, 2 equals week 3, etc.). The final seven columns represent the cumulative daily requirements. Each entry showing an increase from the previous day translates into a cargo generation node in the network.

EDAR KNGU	0	.24	.48	.72	.96	1.20	1.44	1.68
EDAR LGIR	0	.30	.59	.89	1.19	1.48	1.78	2.08
EDAR LIRN	0	.18	.37	.55	.73	.92	1.10	1.28
EDAR OEDR	0	.85	1.69	2.54	3.39	4.23	5.08	5.93
EGUN KNGU	0	.78	1.56	2.34	3.12	3.90	4.68	5.46
EGUN LTAG	0	1.68	3.36	5.04	6.72	8.40	10.08	11.76
KCHS EDAF	0	.16	.20	.22	.46	.75	1.01	1.24
KDOV LGIR	0	.31	.37	.37	.73	1.15	1.64	2.12
KDOV LIPA	0	6.24	7.32	7.50	14.65	23.05	32.91	42.58
KDOV OEDR	0	6.26	7.35	7.53	14.7	23.14	33.04	42.75
KNGU LIPA	0	1.19	1.74	2.01	3.95	6.00	8.32	10.50
KTIK LIPA	0	.51	.77	.91	1.45	2.30	3.12	3.94
KTIK LTAG	0	.83	1.24	1.47	2.35	3.73	5.06	6.39
KTIK OEDR	0	.94	1.41	1.67	2.65	4.22	5.72	7.23
KTIK OERY	0	.50	.75	.89	1.42	2.26	3.07	3.87
LETO KDOV	0	8.19	16.37	24.56	32.75	40.93	49.12	57.31
LETO KTIK	0	.77	1.54	2.31	3.08	3.85	4.62	5.39
LETO KWRI	0	1.16	2.32	3.48	4.64	5.80	6.96	8.12
LETO LERT	0	.60	1.19	1.79	2.39	2.98	3.58	4.18
LETO LIRN	0	.88	1.77	2.65	3.53	4.42	5.30	6.18

Appendix I: Initial Schedule for the E/SWA Sub-Problem

This appendix is an extract of the file "schedule.dat", the initial schedule generated by "makesked.f". "Schedule.dat" is used within "iterate.f" to generate the mission airbase nodes in the network. The first line indicates that there are 213 missions in the mission set. The rest of the file consists of 213 sub-blocks, each representing one mission. The first line of the sub-block shows the route number of the mission, the number of nodes on the mission, the aircraft flying the mission, the capacity of the aircraft, and the type of channel mission (C = requirements, F = frequency of visit). The remaining lines of the sub-block show the ICAO designation of the airbase and its time (day) within the planning horizon.

```
213
056     12      C005    54.0    C
KSUU    3.40000
KTIK    3.52125
KTIK    3.69833
KDOV    3.81554
KDOV    4.57596
EDAF    4.90737
EDAF    5.66779
KDOV    6.06792
KDOV    6.82833
KTIK    6.96171
KTIK    7.13879
KSUU    7.28025
058     12      C005    54.0    C
KSUU    4.50000
KTIK    4.62125
KTIK    4.79833
KDOV    4.91554
KDOV    5.67596
EDAR    5.99525
EDAR    6.75567
KDOV    7.13963
KDOV    7.90004
KTIK    8.03342
KTIK    8.21050
    .
    .
    .
```

Appendix J: User-defined Parameters

This appendix contains a sample of input file "param.dat", containing the parameters which the user can change to enhance program performance. Following the sample file is a discussion of the range of values the parameters may take.

```
100     MAXIT
25      MAXALT
10      MAXTRN
1       CARCRI
2       SORCRI
10      PASSES
2       TRNSHIP
0.001   EPSILON
0.0001  TIME EPSILON
```

MAXIT is the maximum number of iterations through the iterative improvement algorithm per run of the program. The minimum value of this parameter is 1.

MAXALT is the maximum number of alternate paths considered by the shortest path algorithm. Its minimum value is 0.

MAXTRN is the maximum number of transshipments allowed for a single piece of cargo. Its minimum value is 0. While it has no upper bound, tests run with the E/SWA sub-problem indicate that 10 is a realistic maximum.

CARCRI is the cargo priority, which determines the order in which cargo is flowed. Acceptable values are 1 = default, 2 = FIFO, and 3 = largest to smallest.

SORCRI is the mission set sorting criteria, which determines the order in which the schedule improvement algorithm examines the mission set. Acceptable values are 1 = default, 2 = reverse of the order provided, 3 = descending order according to mission utilization, and 4 = ascending order according to mission utilization.

PASSES is the maximum number of passes per iteration of the schedule improvement algorithm. Its minimum value is 1. While it has no upper bound, tests run with the E/SWA sub-problem indicate that 10 is a reasonable maximum.

TRNSHP is the transshipment policy. Acceptable values are 0 = no transshipments allowed, 1 = transshipments are allowed only at a pre-determined list of airbases, and 2 = transshipments may occur at any airbase.

EPSILON and TIME EPSILON are used to prevent problems due to floating point arithmetic. *It is suggested that these remain unchanged.*

Appendix K: Approved Transshipment Bases for the E/SWA Sub-Problem

This appendix contains the file "trnbases.dat". One of the options for the user-defined parameter setting transshipment policy is to allow transshipments to occur only at pre-determined airbases. These airbases may be the only ones within the channel system equipped to handle transshipments. The following is a list of those airbases for the E/SWA sub-problem. The number on line 1 is the total number of approved transshipment bases followed by their ICAO designations.

```
18
EDAF
EDAR
EGUN
KCHS
KDOV
KNGU
KSUU
LERT
LETO
LGIR
LICZ
LIPA
LIRN
LLBG
LTAG
OEDR
OERY
OJAF
```

Appendix L: Mission Utilization Output

This appendix is an extract of an output file "postxxx.c", containing mission utilization information. For each mission a block is created showing the mission number, the route and type of aircraft assigned to the mission, and the aircraft's capacity. Following this information is the ordered list of airbases on the mission, with the utilization of the aircraft on the flight into the airbase and the amount of flow on that flight. The utilization is the percentage of that leg's capacity which was used. At the end of each mission block is a line showing the overall mission utilization, which is a weighted value incorporating the individual legs' utilizations.

An output file like this is generated after every execution of the post-processing subroutine (POSTPROC), numbered according to the iteration. For example, on the first iteration, POSTPROC creates "post001.c".

UTILIZATION OF MISSIONS

UTILIZATION EQUALS THE PERCENTAGE OF A MISSION LEG
CAPACITY THAT IS USED.

ICAO UTIL FLOW

---- ---- ----

MISSION 1 (ROUTE = 56, ACFT = C005, CAPACITY = 54.0 TONS)
KSUU ---- ----
KTIK 0.00 0.00
KDOV 0.00 0.00
EDAF 0.00 0.00
KDOV 0.00 0.00
KTIK 0.00 0.00
KSUU 0.00 0.00
OVERALL UTILIZATION ON THIS MISSION: 0.

MISSION 2 (ROUTE = 58, ACFT = C005, CAPACITY = 54.0 TONS)
KSUU ---- ----
KTIK 0.00 0.00
KDOV 0.03 1.38
EDAR 0.00 0.00
KDOV 0.00 0.00
KTIK 0.00 0.00
KSUU 0.00 0.00
OVERALL UTILIZATION ON THIS MISSION: 2.46219E-03

.
.
.

Appendix M: Sample Flow Pattern Output

This appendix is an extract of flow pattern output generated by the schedule improvement subroutine within "iterate.f". An output file containing this information is created after every execution of the flow and schedule improvement subroutines, numbered according to the iteration. For example, on the first iteration the flow subroutine creates "cflow001.c" and the schedule improvement algorithm creates "paths001.c".

For each piece of cargo flowed a block is created showing that piece's time-in-system (TIS), the amount of the flow (FLOW), the cost of the flow (WTIS), the number of transshipments along the path, and the path it used. The path information contains the ICAO designation of the node, the mission on which that node appears, the time associated with the node, and the node number.

Following the block for the last piece of flowed cargo are cumulative totals for the amount flowed, the CWTIS, and the amount of cargo reflowed in the schedule improvement subroutine (this only appears in output created by the schedule improvement subroutine). These are followed by distributions for the number of transshipments and time-in-system.

TIS: 3.54801 FLOW: 0.240000
COST OF THIS FLOW: 0.851522
NO. OF TRANSSHIPMENTS: 1

ICAO	MSN NO.	TIME	NODE NUMBER
EDAR	0	0.00	1
EDAR	22	2.31	338
EDAF	22	2.34	339
EDAF	22	3.06	340
KCHS	22	3.50	341
KCHS	44	3.50	726
KNGU	44	3.55	727
KNGU	0	0.00	2282

.
.
.

TIS: 0.726650 FLOW: 0.880000
COST OF THIS FLOW: 0.639452
NO. OF TRANSSHIPMENTS: 1

ICAO	MSN NO.	TIME	NODE NUMBER
LETO	0	3.00	136
LETO	102	3.26	1334
EDAF	102	3.37	1335
EDAF	20	3.64	314
LIRN	20	3.73	315
LIRN	0	0.00	2284

TOTAL CARGO FLOWED (TONS): 191.000
TOTAL COST OF THIS FLOW: 297.106
TOTAL CARGO REFLOWED (TONS): 6.37000

TRANSSHIPMENT DISTRIBUTION

NUMBER	OCCURENCES
0	6
1	94

TIME-IN-SYSTEM DISTRIBUTION

DAYS	TONNAGE
0- 1	29.29
1- 2	125.45
2- 3	26.74
3- 4	6.32
4- 5	3.20

Appendix N: Results of Each Iteration

This appendix is the file "run.c", a log file which is created every time the iterative improvement algorithm is run. At the beginning of the file is a listing of the parameters used in the run. Then, for each iteration, the following are presented for the flow and schedule improvement algorithms: 1) a transshipment distribution for the paths used in the flow, 2) a TIS distribution for the cargo which was flowed, 3) the total amount of cargo flowed, and 4) the CWTIS of the flow pattern. The amount of cargo flowed and the CWTIS are displayed *immediately* after the TIS distribution. Note that CARGFLOW refers to the cargo flow algorithm and MODMSN refers to the schedule improvement algorithm.

Parameters:

1	MAXIT
25	MAXALT
1	MAXTRN
1	CARCRI
1	SORCRI
1	PASSES
2	TRNSHIP
0.001	EPSILON
.0001	TIME EPSILON

CARGFLOW CALLED

TRANSSHIPMENT DISTRIBUTION

NUMBER	OCCURENCES
0	7
1	93

T.I.S. DISTRIBUTION

DAYS	TONS
0- 1	20.2800
1- 2	116.280
2- 3	29.9700
3- 4	18.5700
4- 5	3.75000
5- 6	1.31000
6- 7	0.840000
191.000	362.439

MODMSN CALLED
TOTAL CARGO REFLOWED (TONS): 6.37000

TRANSSHIPMENT DISTRIBUTION

NUMBER	OCCURENCES
0	6
1	94

TIME-IN-SYSTEM DISTRIBUTION

DAYS	TONNAGE
0- 1	29.29
1- 2	125.45
2- 3	26.74
3- 4	6.32
4- 5	3.20
191.000	297.106

ITERATION 1 COMPLETED.

CARGFLOW CALLED

TRANSSHIPMENT DISTRIBUTION

NUMBER	OCCURENCES
0	6
1	98

T.I.S. DISTRIBUTION

DAYS	TONS
0- 1	40.0400
1- 2	119.860
2- 3	30.2900
3- 4	6.08000
4- 5	5.40000
201.670	310.503

Appendix O: Test Runs

This appendix contains a complete list of the runs used for testing.

Run #	MAXTRN	CARCRI	SORCRI	PASSES	REFLOW	Final Flow	Final CWTIS
1	1	1	1	10	on	211.93	247.32
2	2	1	1	10	on	226.93	254.98
3	3	1	1	10	on	229.99	258.90
4	10	1	1	10	on	229.99	232.79
5	1	1	2	10	on	203.19	238.13
6	2	1	2	10	on	225.48	265.60
7	3	1	2	10	on	229.99	241.17
8	10	1	2	10	on	229.99	231.44
9	1	1	3	10	on	202.78	232.49
10	2	1	3	10	on	226.45	281.74
11	3	1	3	10	on	229.99	261.48
12	10	1	3	10	on	229.99	257.18
13	1	1	4	10	on	212.07	241.72
14	2	1	4	10	on	225.47	252.15
15	3	1	4	10	on	229.99	242.30
16	10	1	4	10	on	229.99	235.49
17	1	1	1	10	off	191.00	260.62
18	2	1	1	10	off	227.34	310.97
19	3	1	1	10	off	229.99	284.89
20	10	1	1	10	off	229.99	284.89
21	1	1	2	10	off	191..00	260.62
22	2	1	2	10	off	227.34	310.97
23	3	1	2	10	off	229.99	284.89
24	10	1	2	10	off	229.99	284.89
25	1	1	3	10	off	191.00	260.62
26	2	1	3	10	off	227.34	310.97
27	3	1	3	10	off	229.99	284.89
28	10	1	3	10	off	229.99	284.89
29	1	1	4	10	off	191.00	260.62
30	2	1	4	10	off	227.34	310.97
31	3	1	4	10	off	229.99	284.89
32	10	1	4	10	off	229.99	284.89
33	10	1	1	10	on	2115.1	18151.0
34	10	2	1	10	on	2298.9	21974.0
35	10	3	1	10	on	2227.7	20170.0

In all test runs, the following parameters remained constant:

MAXIT = 100
MAXALT = 25
TRNSHP = 2
EPSILON = .001
TIME EPSILON = .0001

Recall that:

MAXTRN = the maximum number of transshipments allowed per piece of cargo flowed
CARCRI = the cargo flow priority
SORCRI = the mission set sorting criteria
PASSES = the number of passes per iteration of the schedule improvement algorithm
REFLOW = reflow mechanism
TRNSHP = the transshipment policy

Appendix P: Output Conversion Subroutine

This appendix is the FORTRAN program "makeraw.f", which converts the schedule created by "iterate.f" back into the format required for validation by CARGOSIM.

```
      PROGRAM MAKERAW

*     THIS PROGRAM TAKES THE SCHEDULE CREATED BY THE SCHEDULE
*     IMPROVEMENT ALGORITHM IN THE IIA AND CONVERTS IT BACK INTO THE
*     FORMAT OF 'schedule.raw' FOR VALIDATION IN CARGOSIM

      REAL TIME
      INTEGER ROUTE,NODES
      CHARACTER*4 ACFT,ICAO

     OPEN(UNIT=1,FILE='newsched.dat',STATUS='OLD',FORM='FORMATTED')
     OPEN(UNIT=2,FILE='newsched.raw',STATUS='UNKNOWN',
    c                                                FORM='FORMATTED')

      READ(1,*)

5     READ(1,*,END=30)ROUTE,NODES,ACFT
      READ(1,*,END=30)ICAO,TIME
      WRITE(2,10)ROUTE,ACFT,TIME
10    FORMAT(I3,3X,A4,3X,F7.4)

      DO 20 I=1,NODES-1
       READ(1,*,END=30)
20    CONTINUE

      GOTO 5

30    WRITE(*,*)'PROGRAM COMPLETED'

      END
```

Bibliography

Ackley, M., W. Carter, G. Hughes, J. Litko, K. Ware, A. Whisman, and R. Roehrkasse. *Optimization Applications at the Military Airlift Command: Importance and Difficulties*. Unpublished paper provided to the Second International Conference on Industrial and Applied Mathematics, Washington D.C., July 8-12, 1991.

Bazaraa, Mokhtar S., John J. Jarvis and Hanif D. Sherali. *Linear Programming and Network Flows*. New York: John Wiley & Sons, 1990.

Bodin, Lawrence D., Bruce Golden, Ariang Assad and Michael Ball. "Routing and Scheduling of Vehicles and Crews: The State of the Art," *Computers in Operations Research*, 10: 63-209 (1983).

Bodin, Lawrence D. "Twenty Years of Routing and Scheduling," *Operations Research*, 39: 571-579 (July-August 1990).

Borsi, MAJ John. Account of a personal interview with Captain Michael Del Rosario, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 6 August 1992.

Borsi, MAJ John. Personal interview. Air Force Institute of Technology, Wright-Patterson AFB OH. 15 July 1993.

Borsi, MAJ John. Personal interview. Air Force Institute of Technology, Wright-Patterson AFB OH. 23 July 1993.

Borsi, MAJ John. Personal interview. Air Force Institute of Technology, Wright-Patterson AFB OH. 4 February 1994.

Carter, Brand and Joseph R. Litko. *Simulating the Air Mobility Command Channel Cargo System*. Unpublished paper provided by Lt. Jonathan Robinson, Command Analysis Group, Air Mobility Command, HQ AMC/XPYR, Scott AFB, IL.

Del Rosario, Capt Michael. *Determining Cargo Flow for Air Mobility Command's Channel Cargo System*. MS thesis, AFIT/GOR/ENS/93M-04. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1993.

"A Description of the STORM Linear Programming Problem for Analysis of Channel Cargo Routing." AMC working paper provided by 1LT J. Robinson, Command Analysis Group, Air Mobility Command, HQ AMC/XPYR, Scott AFB, IL. 20 September 1993.

DoD Materiel Management Regulation (DoD 4140.1-R). Office of the Assistant Secretary of Defense (Production & Logistics).

Harel, David. *Algorithmics: The Spirit of Computing*. Massachusetts: Addison-Wesley Publishing Company, 1992.

Lin, S. "Heuristic Programming as an Aid to Network Design," *Networks*, 5: 33-43 (1975).

Moul, Capt Justin E. *A Method for Determining Schedule Delay Information in a Channel Cargo Route Network Schedule*. MS thesis, AFIT/GST/ENS/92M-05. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, June 1992.

Rau, Capt Gregory S. *Scheduling Air Mobility Command's Channel Cargo Missions*. MS thesis, AFIT/GOR/ENS/93M-19. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1993.

Robinson, 1LT Jonathan. Personal interview and data derived from various AMC studies. Command Analysis Group, Air Mobility Command, HQ AMC/XPYR, Scott AFB, IL. 20 September 1993.

Robinson, 1LT Jonathan. Personal interview. Command Analysis Group, Air Mobility Command, HQ AMC/XPYR, Scott AFB, IL. 1 February 1994.

Shepherd, Capt Dave. "Peacetime Airlift: Job #1, Too!" *Defense Transportation Journal*, Vol. 46: 11-19 (1990).

Syslo, Maciej M., Narsingh Deo, and Janusz S. Kowalik. *Discrete Optimization Algorithms with Pascal Programs*. New Jersey: Prentice-Hall, Inc., 1983.

Whisman, Alan W. *Channel Routing Model*. Unpublished paper provided by Lt. Jonathan Robinson, Command Analysis Group, Air Mobility Command, HQ AMC/XPYR, Scott AFB, IL.

Zanakis, Stelios H. and James R. Evans. "Heuristic 'Optimization': Why, When, and How to Use It," *Interfaces*: 84-91 (October 1981).

Zanakis, Stelios H., James R. Evans and Alkis A. Vazacopoulos. "Heuristic Methods and Applications: A Categorized Survey," *European Journal of Operational Research*, 43: 88-110 (1989).

Vita

Captain John Fitzsimmons Jr. was born 11 August 1966 in West Allis, Wisconsin. He graduated from Hartford Union High School in 1984 and attended the United States Air Force Academy, graduating with a Bachelor of Science (specialty: Physics) in June 1988. He was assigned as a scientific analyst in the 86th Fighter Weapons Squadron at Eglin AFB, Florida. In this position, he performed battle damage analysis on simulated targets, debriefed tactical aircrews, evaluated a variety of air-to-ground precision guided munitions, maintained several computer databases, and acted as the project analyst on annual reports. After this assignment, he entered the Air Force Institute of Technology in August 1992 to pursue a Masters of Science in Operations Research. Following graduation, Captain Fitzsimmons is to be assigned to the 57th Test Group, Nellis AFB, Nevada.

Permanent Address: 7243 Roosevelt Rd.
Hartford, Wisconsin 53027

Vita

Captain John Walker was born on 6 September 1966 in Beckley, West Virginia. He graduated from Oceana High School in 1984 and attended West Virginia University, graduating in December 1988 with a Bachelor of Arts degree in Mathematics. Commissioned through AFROTC in January 1989, Captain Walker was assigned as an analyst with the Command Analysis Group, HQ Military Airlift Command (Air Mobility Command), Scott AFB, IL, where he served from March 1989 to August 1992. He entered the School of Engineering, Air Force Institute of Technology, in August 1992. Following graduation, Captain Walker will be assigned to the Air Force Personnel Operations Agency at the Pentagon.

Permanent address: P.O. Box 247
Kopperston, WV 24854

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1994	Master's Thesis

4. TITLE AND SUBTITLE
A HEURISTIC APPROACH TO DETERMINING CARGO FLOW AND SCHEDULING FOR AIR MOBILITY COMMAND'S CHANNEL CARGO SYSTEM

5. FUNDING NUMBERS

6. AUTHOR(S)
John D. Fitzsimmons Jr., Captain, U.S. Air Force
John M. Walker, Captain, U.S. Air Force

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology,
WPAFB, OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GOR/ENS/94M-05

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

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This research investigated a heuristic approach to schedule aircraft for the channel cargo system of the United States Air Force's Air Mobility Command (AMC). Given cargo/frequency of visit requirements, a fleet of aircraft, and possible routes, the objective of this research was to develop, implement, and test an iterative procedure to efficiently schedule and load aircraft in order to maximize the flow of cargo through the channel cargo system. Once a level of flow was established, attempts were made to minimize cost in terms of cumulative weighted time-in-system (CWTIS). A minimum cost flow heuristic, incorporating a successive shortest path algorithm, was coupled with a critical arc schedule improvement heuristic. Our procedure iterated between these two heuristics to generate a cargo flow pattern and aircraft schedule. This research demonstrated the usefulness and efficiency of this heuristic in planning airlift for the channel cargo system. The FORTRAN programs which implement the heuristics are compatible with current AMC scheduling/advance planning tools. Given this compatibility, additional testing in conjunction with AMC's current planning tools (STORM, CARGPREP, and CARGOSIM) is warranted. Pending successful testing in this environment, implementation of these methods is recommended.

14. SUBJECT TERMS
Heuristics, Channel Cargo System, Networks, Schedule, Multicommodity, Shortest-Path, Interchange, Flow Pattern

15. NUMBER OF PAGES
148

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL