THE BOTTLENECK ROUTING OF MILITARY CARGO AIRCRAFT

by

Murat Kasaroğlu

B.S. in I.E., Boğaziçi University, 1980

Submitted to the Institute for Graduate Studies in Science and Engineering in partial fulfillment of the requirements for the degree of

Master

of

Science

Boğaziçi University

1983

We hereby recommend that the thesis entitled "The Bottleneck Routing of Military Cargo Aircraft" submitted by Murat Kasaroğlu be accepted in partial fulfillment of the requirements for the Degree of Master of Science in Industrial Engineering in the Institute for Graduate Studies in Science and Engineering, Boğaziçi University.

EXAMINING COMMITTEE

Doç. Dr. Gündüz ULUSOY
(Thesis Advisor) [illegible]

Y.Doç. Dr. Ahmet ALKAN [illegible]

Y.Doç. Dr. Süleyman ÖZEKİCİ [illegible]

Date: 28/4/1983

178216

ACKNOWLEDGEMENTS

I am indepted to Doç. Dr. Gündüz Ulusoy, the Chairman of my Supervisory Committee, for his valuable guidance, help and support throughout the course of the study.

I would like to express my sincere gratitude to Yarbay Mehmet Tuzcu for his helpful suggestions and comments.

I would also like to thank my wife who has cheerfully and patiently encouraged my study.

Murat Kasaroğlu

ABSTRACT

The objective of this study is to develop an efficient method of routing military cargo planes in war time.

A model with a nonlinear objective function is developed to determine these routes which minimizes the maximum mission time of planes. Based on this model, a solution procedure is introduced. Then, a heuristic procedure is suggested to handle various airport capacity constraints. As an extension of the study, a proposal is made on a model which minimizes the total mission of the planes.

Consequently a study is made on the behaviour of the model on some special cases.

ÖZET

Bu çalışmanın amacı, askeri yük uçaklarının savaş sırasındaki güzergâhlarını belirleyecek bir metod geliştirmektir.

Bu güzergâhları belirleyebilmek için en uzun görev süresini enküçükleyecek ve yaddoğrusal amaç işlevli bir model geliştirilmiştir. Bu modele dayandırılarak bir çözüm yordamı tanıtılmıştır. Daha sonra havaalanlarındaki çeşitli kısıtları göz önüne alan bulgusal bir yordam sunulmuştur. Çalışmanın bir uzantısı olarak ise toplam görev süresini enküçülten bir model öne sürülmüştür.

Çalışma, geliştirilen modelin bazı özel şartlar altındaki davranışını inceleyerek sonuçlandırılmıştır.

ASKERİ UÇAKLARIN İNTİKAL PLANLAMASI

Bu çalışmanın amacı, askeri yük uçaklarının savaş sırasındaki güzergâhlarını belirleyecek bir metod geliştirmektir.

Bu güzergâhları belirleyebilmek için en uzun görev süresini enküçükleyecek ve yaddoğrusal amaç işlevli bir model geliştirilmiştir. Bu modele dayandırılarak bir çözüm yordamı tanıtılmıştır. Daha sonra havaalanlarındaki çeşitli kısıtları göz önüne alan bulgusal bir yordam sunulmuştur. Çalışmanın bir uzantısı olarak ise toplam görev süresini enküçülten bir model öne sürülmüştür.

Çalışma, geliştirilen modelin bazı özel şartlar altındaki davranışlarını inceleyerek sonuçlandırılmıştır.

THE BOTTLENECK ROUTING OF MILITARY CARGO AIRCRAFT

The objective of this study is to develop an efficient method of routing military cargo planes in war time.

A model with a nonlinear objective function is developed to determine these routes which minimizes the maximum mission time of planes. Based on this model, a solution procedure is introduced. Then, a heuristic procedure is suggested to handle various airport capacity constraints. As an extension of the study, a proposal is made on a model which minimizes the total mission of the planes.

Consequently a study is made on the behaviour of the model on some special cases.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

1.1 THE DESCRIPTION OF THE PROBLEM

The airforce always needs an efficient method of defining routes and schedules of military cargo planes both in war and in peace time. Several military airports are located throughout the country with some of them being the bases of those cargo planes and there are several loads that have to be carried among those airports. The basic unit of shipment is one plane load. In peace time, the routing and scheduling process is done periodically. That is, demands between airports are generated within a period and these demands are satisfied within the next period. In case of a war, we think of a one-time operation where cargo planes located at known bases carry equipment and military personnel in known quantities from supply points to demand points. Without loss of generality, all demand and supply nodes are assumed to be the airports. Since swiftness is an essential ingredient for success in a war, the overall job should be completed as fast as possible. That is, the longest mission time of the planes should be minimized. For peace time operations, on the other hand, the total mission time of all planes should be minimized, since the variable cost of transportation is assumed to be directly proportional to distance.

Also there are constraints on servicing planes at the airports. Each airport has a given service capacity for loading and unloading of the planes at any time. This handling capacity constraint should be considered simultaneously with the queue capacity of each airport. When the handling capacity of an airport is exceeded, then the excess planes should join the queue at this airport. But some airports cannot hold more than a given amount of queue, mainly because there are no available parking space for these planes within that area. But a more important reason is that, in war time it is not recommended practice to allow for the accumulation of airplanes above a given number at any time, since the enemy can attack any one of these airports at any time. So the queue lengths at those ports should not exceed certain prespecified levels. Therefore, a schedule satisfying these constraints besides minimizing the mission time is required.

The problem described above falls within the class of problems called "the vehicle routing and scheduling problems" in literature. A brief review of these problems will be given in the next section.

1.2 THE CURRENT STATE OF ART IN VEHICLE ROUTING AND SCHEDULING

The routing and scheduling of vehicles and crews is an area of both theoretical and practical importance to both operations researchers and transportation planners. Recently, significant progress has been made in the problem formulations and in the design, analysis, and implementation of solution procedures.

From a practical point of view, the effective routing and scheduling of vehicles and crews can save the state and private enterprises many millions of TL's a year. In addition, these routing and scheduling procedures can increase productivity, improve operations, aid in long-range planning, assist contract negotiations, make the job of the scheduler or dispatcher much easier to handle, and help to control the financial impact of adverse weather conditions on vehicle utilization.

The Vehicle Routing Problem (VRP) can be stated as follows: Given a set of nodes (points) and/or arcs to be serviced by a fleet of vehicles, find the routes of each vehicle so that total time and/or total cost of transportation is minimum. A vehicle route is a sequence of pickup and/or delivery points which the vehicles must traverse in order, starting and ending at a depot or domicile.

Above statement is only one definition of vehicle routing problem. The problem has many extensions to suit the practical problem addressed.

Vehicle Routing Problems can be classified as node routing problems, arc routing problems, and general routing problems. The problem of visiting all nodes in a network and returning to the starting point (node routing) while incurring minimal cost is the Travelling Salesman Problem (TSP). In node routing problems a collection of origin/destination pairs of nodes are given and at least one vehicle must travel from each origin to its corresponding destination. Examples of this problem are newspaper delivery and dial-a-ride or messenger service. The problem of covering all arcs in a network while minimizing total distance travelled (arc routing) is the Chinese Postman Problem (CPP). In arc routing problems, a collection of arcs in a network has to be covered. Examples of this problem are snow removal and street sweeping. The General Routing Problem is a generalization which includes both TSP and CPP as special cases. Here we seek the minimum cost cycle which visits every prespecified node and arc. Examples of such problems are school bus routing and household refuse collection. The generic problems such as TSP and CPP are not of practical interest, but of value for solving VRP and gaining insight.

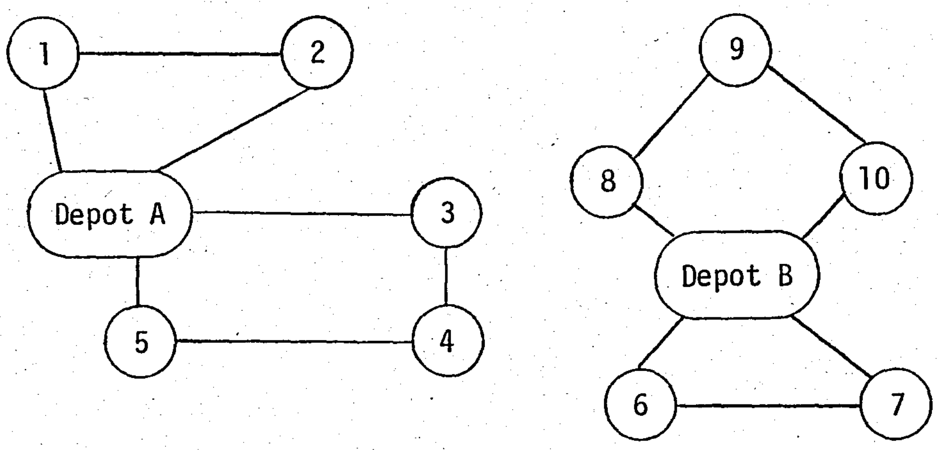
In general, node routing problems require a set of delivery routes from a central depot(s) to demand points, each having known or stochastic requirements, in order to minimize the total distance covered by the entire fleet. Vehicles have known capacities and possibly maximum route time constraints. All vehicles start and finish the job at specific depot(s).

A set of vehicle routes that service 10 demand points are shown in Fig. 1.1. Each node has demand of unity and each vehicle has a capacity of three units.

Bodin, Golden, and Assad (1981) have summarized various studies on extension of routing problems in three classes as follows,

i. "one-to-many" problems:

Such problems have a central depot and many destinations. Items are loaded on the vehicles at the depot and delivered to many destinations.



Route 1:	Depot A-1-2-Depot A
Route 2:	Depot A-3-4-5-Depot A
Route 3:	Depot B-6-7-Depot B
Route 4:	Depot B-8-9-10-Depot B

FIGURE 1.1 - Illustration of Routes

ii. "many-to-one" problems:

Also in this case there is a central depot, and many pickup points. Items are collected from these points and delivered to that central depot.

iii. "many-to-many" problems:

Each item to be serviced can have a different pickup point (origin) and a different delivery point (destination).

Most of the time authors talk about the pickup and delivery locations of items being serviced in "many-to-many" problems and do not explicitly worry about the garages where the vehicles are stationed. The deadhead times to go from the depots to the garages (or the times from garages to the first stop on the routes and the times to the garages from the last stop on the routes) are generally added to the length of the routes after the routes are formed and not considered a part of the optimization. For many problems, this is a fixed time since there is only one garage that can house the vehicles (out of the depot); in other cases, the routes might be altered somewhat if this distance to and from the garage were taken into account in the optimization.

The Vehicle Scheduling Problems (VSP) can be stated as routing problems with additional constraints on times of performing activities. Each location may require delivery within an interval. Thus the movements of vehicles should be followed both in space and time. A vehicle schedule is a sequence of pickup and/or delivery points together with an associated set of arrival and departure times. The vehicle must traverse the points in the designated order and at the specified time intervals.

When arrival times at the nodes ard/or arcs are fixed in advance we refer to the problem as a scheduling problem. When the arrival times are unspecified, then the problem is a straight forward routing problem.

When time windows and/or precedence relationship exist so that both routing and scheduling functions need to be performed, we view the problem as a combined routing and scheduling problem. The combined routing and scheduling problems often arise in practice and representatives of many real-world applications (Boding and Golden, 1981).

Bodin, Golden and Assad (1981) have described some examples related with this topic as follows,

i. School Bus Routing and Scheduling

There are a number of schools and each one has a set of bus stops associated with it. In addition, there is a given number of students associated with each bus stop. Each school has a fixed starting time and a fixed ending time with corresponding time windows for school bus routing. The time window before the starting time of the school involves the time window for the delivery of students to the school in themorning and the time window after the ending time of the school in the afternoon is the time window associated with the pickup of the students. The principle objective when utilizing a leased fleet of vehicles is to minimize the number of buses required while servicing all the students and satisfying all the time windows. When operating a fleet owned by the district, the objective is to minimize a combination of transportation costs and the number of vehicles used.

Although most papers related with this topic focus primarily on the routing component, Bodin and Berman (1979) suggested a procedure for forming daily bus schedules as well as methods for routing buses. The routing component of their suggestion forms a set of routes for each school. Each route is feasible with respect to the maximum available time for the students and the maximum capacity of buses. The scheduling component organizes the partial routes for each of the schools into daily schedules for the buses.

ii. Tractor-Trailer Routing and Scheduling with Full Loads

A common commercial distribution problem is the routing and scheduling of tractors or tractor trailer front ends with full loads. The term full load means that a trailer is attached to the tractor and has to be transported from a pickup point (the origin) to a delivery point (the destination). The load of a trailer has a unique destination and is not to be split among different destination locations. The capacity of a tractor is one trailer. Since each trailer is transported from its origin to its destination, the trailer problem obviously involves precedence constraints.

The demands are specified in terms of the number of trailer trips between origin/destination pairs. Given this demand data, one may address the following two decision problems:

a) Minimize the total distribution cost for handling all origin-destination demans.

b) Determine the optimal fleet size required to service a subset of the origin destination demands given that the remaining demand is to be serviced by common carrier.

Love (1978) suggested a model involving two submodels essentially for the solution of this problem. One of the submodels is the tractor submodel and the other is trailer submodel.

iii. Tractor-Trailer Routing and Scheduling with Partial Loads

This problem is similar to the full load problem except that each origin-destination pair need not to have a full trailer load to be serviced. Consequently, the load on a trailer may be split among different destinations.

iv. Street Sweeper and Household Refuse Collection Routing and Scheduling

The problems of scheduling street sweepers and household refuse collection vehicles are applications of the Chinese Postman Problem. For both of these problems, a set of street segments is specified as needing service. The problem is to arrange a set of tours (each tour corresponding to a vehicle) covering all such segments that minimizes the number of vehicles used. A surrogate but highly correlated objective is to minimize the total deadhead time of the vehicles. There are no precedence relationships on the entities to be serviced, and the time windows correspond to the parking regulations.

Golden and Wong (1981) showed how capacitated arc routing formulations can be applied to these problems.

v. Airplane Scheduling

The scheduling of airplanes for commercial airlines is a very complicated procedure and is embedded within the process of generating a time table for the airline. The generation of a time table has to take into account such factors as the expected number of passengers travelling between cities, frequency of service desired, nonstop versus multiple stop service, etc. Furthermore, this scheduling takes into account the problems of generating pairings and bid lines for the crews. Thus, airlines may change their time table and plane schedules if a pairing can be saved. At this time, most scheduling of airplanes for commercial airlines is carried out on a manual basis or in an interactive computing mode and little algorithmic sophistication is utilized in the process.

Soumis, Ferland and Rousseau (1981) and Richardson (1975) had given mixed integer programming formulations for both of the plane and passenger sides of the problem.

vi. Dial-A-Ride Routing and Scheduling Problems

In recent years, the area of dial-a-ride routing and scheduling has received considerable attention. In the dial-a-ride problem, customers call in to request service. Each customer specifies a distinct pickup and delivery point and, perhaps, a desired time for pickup or delivery. If all customers demand immediate service, then routing and scheduling is done in real time and the problem is referred to as the dynamic or real time dial-a-ride problem. If all customers call in advance, so that a complete data base of customer demand is known before any routing or scheduling is carried out, then this problem is referred to as the subscriber or static dial-a-ride problem. Both dynamic and static dial-a-ride problems have precedence relationships since a customer must be pickep up before he is delivered. In some situations a desired time of pickup or delivery is specified in advance and the "other service" (either delivery or pickup) must be carried out within a given number of minutes from either the desired or the actual time of delivery or pickup. In a certain sense this introduces a two-sided time window on the "other service".

Psaraftis (1980) and Stein, et.al (1978) have given formulations to various derivatives of the dial-a-ride problem.

1.3 COMPLEXITY OF VEHICLE ROUTING AND SCHEDULING PROBLEMS

All of the problems mentioned in the previous section are NP-hard. Moreover, the complications in these problems are such that exact algorithmic approaches based on mathematical programming formulations have not been successful for these problems.

The network problems are classified according to a theoretical scheme based on the notions of "polynomially-bounded" and "NP-hard" as follows. The polynomially-bounded class P is composed of such problems for which polynomially-bounded algorithms are known. An algorithm is said to run in polynomial time if there exists an upper bound on the number of operations, that is a polynomial in n, where n is an input parameter which measures the problem size (such as the number of nodes). Thus the computational effort increases only polynomially with problem size in the worst case. The problems of this class can generally be solved quite efficiently and their order is determined by the highest power of n in polynomial expression.

But on the other hand, there is a large class of network and combinatorial problems for which no polynomially-bounded algorithm exists. Such problems are called NP-hard (NP stands for nondeterministic polynomial). The solution procedures developed for such problems require exponential run time. That is computational effort increases exponentially with the problem size.

To emphasize the difference, assume that there are two algorithms available to solve a network problem with respective run times are:

$$f_1(n) = 1000n^2 \qquad \text{and} \qquad f_2(n) = 2^n$$

where n is the number of nodes. When n is small the second algorithm will work faster than the first one. But if n is increased a little (Let n = 20), then the second algorithm will collapse, although the first one still functions well. The difference between these two algorithms grow even more as computer technology improves. If the efficiency of a computer improves by a factor of 100, then the maximum problem size solvable by first algorithm in a fixed amount of time would increase by a factor of 10 whereas the maximum problem size handled by second algorithm increase by no more than seven nodes.

It is obvious that, an algorithm of order n^2 is preferable to one of n^4, and exponential time algorithms are to be avoided whenever possible (Golden, Ball and Bodin, 1981).

All routing and scheduling problems of interest fall in the class of NP-hard problems. Apparently minor changes in problem characteristics may result in radical changes in the computational complexity of the resulting problems. For example both directed and undirected Chinese Postman Problem are in the class P, whereas Mixed Chinese Postman Problem (where both directed and undirected arcs are allowed) is NP-hard. Table 1.1 reviews some algorithms available for network problems and compares their behaviour. This table was presented at NSF Workshop on Large Scale Systems in Lubbock, Texas in April, 1979.

TABLE 1.1 - Comparison of Different Algorithms

Problem Name	Heuristic Algorithm		Exact Algorithm	
	Size Handled Easily	References	Size Handled Easily	References
Shortest Path from s to t	NN		5000	Golden and Ball (1978)
Shortest Path from s to all other nodes	NN		5000	Denardo and Fox (1979), Golden (1976), Pape (1974), Gilsinn and Witzgall(1973), Dial, et al. (1979).
Shortest Paths between all nodes	NN		500	Relton and Law (1978)
K Shortest Paths	NN		500 (K 5)	Shier (1976), Shier (1979)
Minimal Spanning Tree	NN		5000	Kershenbaum and Van Slyke (1972)
Capacitated Minimal Spanning Tree!	1000	Kershenbaum (1974)	40	Chandy and Lo (1973)
Transportation Problem	NN		3000	Mulvey (1978), Bradley, et al. (1977), Glover, et al. (1974)
Max Flow	NN		3000	Cheung (1980), Glover, et al. (1974)
Min Cost Flow	NN		3000	Bradley, et al. (1977), Barr, et al. (1974)
Matching	NN		500	Cunningham and Marsh(1978), Derigs (1979), Derigs and Kazakidis (1979)

(Table 1.1 continued)

Problem Name	Heuristic Algorithm		Exact Algorithm	
	Size Handled Easily	References	Size Handled Easily	References
Travelling Salesman Problem!	1000	Webb (1971), Golden and Bodin (1978), Golden, et al. (1980)	100	Liliotis (1976), Miliotis (1978), Held and Karp (1970), Padberg and Hong (1977), Balas and Christofides(1981)
Vehicle Routing Problem!	750	Golden, et al. (1977)	30	Christofides, et al. (1981)

! indicates problem is NP-hard.

NN indicates heuristic or approximate algorithms are not necessary.

1.4 THE OUTLINES OF THE MODEL DEVELOPED FOR "ROUTING THE MILITARY CARGO AIRLINES"

The model developed in this thesis treats the problem in two stages. In the first stage, the routes of the planes are determined regardless of the airport capacities. These routes are determined such that all loads are carried to appropriate locations, and the objective function is minimized. The objective function can be stated as follows depending on the nature of the problem:

i. Minimize the maximum job time.
ii. Minimize the total job time.

The first problem is called the "Bottleneck Routing Problem (BRP)" and the other is called the "Minimum Total Time Routing Problem (MTRP)".

At the second stage, the schedules of planes are determined such that the airport capacity constraints are satisfied, with the given routes of the planes.

Nearly in all the problems discussed in the previous sections, one of the major objectives is the minimization of the number of vehicles required, besides minimization of total transportation costs. But in military applications the number of planes that will be utilized is already given in most of the cases and the major objective is the minimization of maximum job time. So, this nature of the objective function does not allow us to utilize any of the solution techniques yet developed for problems with linear objective functions. Except Ulusoy (1981) there is no significant effort on such objective functions in the literature yet.

The military cargo airplane routing problem can be classified as a "many-to-many" routing problem in which each item to be serviced can have different origin and destination points. But the planes are housed in one or more of these origin and/or destination points. So the model takes care of the initial locations of the planes (garages of the vehicles) and all the deadhead times are considered seperately, since there is more than one airport to house planes initially.

Also the problem is subject to precedence constraints, since the plane should be loaded before it is unloaded. But the load on a plane is not splitable. The demand between any origin-destination pair is in terms of full plane loads.

Besides the objective function, the basic difference between military airplane routing problem and classical vehicle routing problem is the definition of demands. In vehicle routing problems, demands are located at nodes and should be supplied from a central depot, but in our case a demand node can be the supply node of another demand node. Hence items should be transferred among them. By this definition of demands, the problem can be viewed as a dial-a-ride problem, but in dial-a-ride problems depot location is known. On the other hand, in our case the depots have the same characteristics of other nodes of the system except they have some planes initially.

Also the problem have similarities with the tractor-trailer routing problem with full loads. In both cases some cargo has to be shipped between prespecified points and the cargo is not splitable among different locations. But in case of military routing there is no distinction such as tractor and trailer.

1.5 SUMMARY OF THE WORK FOLLOWING

Chapter 2 formulates the model, summarizes the solution technique suggested by Ulusoy (1981) and introduces the classical Set Partitioning Method.

Chapter 3 describes the model developed for bottleneck routing of cargo planes.

Chapter 4 introduces an heuristic to handle airport capacity constraints.

Chapter 5 introduces the transformations suggested for minimum total time routing.

II. MATHEMATICAL FORMULATION AND THEORETICAL BACKGROUND

2.1 MATHEMATICAL FORMULATION OF THE BOTTLENECK ROUTING PROBLEM

The Bottleneck Routing Problem (BRP) can be stated as follows:

Given, $L = \{\ell_1, \ell_2, \ldots, \ell_M\}$ be the set of all loads that have to be carried between all airports. Let, $y = \{S_1, \ldots, S_N\}$ be the set of all sets, $S_J \subset L$. Each S_J defines a set of loads that can be carried by a plane. Let for each S_J, be an associated time figure C_J, defining the time required to carry all loads shown by S_J.

Then, a "set-partitioning" $\overline{S}_\ell$ of L, is any subset of y obeying following rules;

i. $$\bigcup_{S_J \varepsilon \overline{S}_\ell} S_J = L \tag{2.1}$$

ii. $$S_J \cap S_m = \emptyset, \quad \forall S_J, S_m \varepsilon \overline{S}_\ell, \qquad m \neq j \tag{2.2}$$

Rule i imposes that all loads should be covered by that set-partitioning of L, and Rule ii imposes that each load should be covered exactly once.

Let the set $\overline{\mathcal{S}}$ is the set of all $\overline{S}_\ell$ (all set-partitionings of L) satisfying the rules given above.

Then, the Bottleneck Routing Problem is to find that set-partitioning of L, in which the maximum C_J value is minimized. That is, the maximum time required to finish each job is minimized.

Thus, (0-1) linear programming formulation of the BRP can be given as:

$$\text{Min } Z = \min_{\overline{S}_\ell \varepsilon \overline{\mathcal{S}}} [\max_{S_J \varepsilon \overline{S}_\ell} (C_J)] \tag{2.3}$$

$$\text{s.t.} \sum_{J=1}^{N} t_{iJ} y_J = 1, \qquad i = 1,2,\ldots,M \tag{2.4}$$

$$\sum_{J=1}^{N} h_{Jk} y_J = H_k, \qquad k = 1,2,\ldots,RP \tag{2.5}$$

$$y_J = \begin{cases} 1, & S_J \varepsilon \overline{S}_\ell \text{ for any } \overline{S}_\ell \varepsilon \overline{\mathcal{S}} \\ 0, & \text{otherwise} \end{cases} \tag{2.6}$$

where,

$$t_{iJ} = \begin{cases} 1, & \text{if load } \ell_i \varepsilon S_J \\ 0, & \text{if load } \ell_i \not\varepsilon S_J, \text{ for } \begin{matrix} i = 1,\ldots,M \\ J = 1,\ldots,N \end{matrix} \end{cases} \tag{2.7}$$

$$h_{Jk} = \begin{cases} 1, & \text{if plane covering } S_J \text{ is originally at airport } k \\ 0, & \text{otherwise, for } \begin{matrix} J = 1,\ldots,N \\ k = 1,\ldots,RP \end{matrix} \end{cases} \tag{2.8}$$

and

$C_J = f(h_{Jk}, S_J)$: is the time spent by a plane which is originally at airport k to cover all loads in S_J. There are many possibilities to cover all loads in S_J. Therefore, C_J must correspond to that of the shortest among such routes. (2.9)

Also,

P :	is the number of planes .
R :	is the number of airports
RP:	is the number of airports which have planes initially
M :	is the number of loads
N :	is the number of sets in Y
H_k:	is the initial number of planes at airport k.

Note that,

$$\sum_{k=1}^{RP} h_{Jk} = 1, \qquad \text{for } J = 1,\ldots,N \tag{2.10}$$

and number of planes is given as:

$$P = \sum_{k=1}^{RP} H_k \tag{2.11}$$

Thus, the first constraint in the formulation (Eq. (2.4)) imposes the so-called "no-overcovering" restriction on the problem. That is, each load should be covered exactly once. The second constraint (Eq. (2.5)) forces the problem to use exactly the prespecified number of planes from each airport.

The mathematical formulation of Minimum Total Time Routing Problem (MTRP) differs in the objective function only. That is,

$$\text{Min } Z = \sum_{J=1}^{N} C_J y_J \tag{2.12}$$

The constraints are (2.4), (2.5), (2.6) as in the previous problem.

2.2 THE SET PARTITIONING PROBLEM

The following two sections about the set partitioning problem are adopted from Chapter 3 of Christofides (1975).

2.2.1 The Problem Formulation

The Set Partitioning Problem (SPP) owes its name to the following set-theoretic interpretation.

Given a set $L = \{\ell_1,\ldots,\ell_M\}$, and a set $\mathcal{S} = \{S_1,\ldots,S_N\}$ of sets $S_J \subset L$, and a subset $\overline{S}_\ell = \{S_{J1}, S_{J2},\ldots,S_{Jp}\}$ of $\mathcal{S}$, such that the rules given in expressions (2.1) and (2.2) define $\overline{S}_\ell$, then $\overline{S}_\ell$ is called a "set partitioning of L". If the second rule is omitted, then $\overline{S}_\ell$ is called a "set-covering of L". That is, when S_J's within $\overline{S}_\ell$ are not pairwise disjoint.

To be consistent with the definition of BRP, define the set $\overline{\mathcal{S}}$ as the set of all $\overline{S}_\ell$. Hence $\overline{\mathcal{S}}$ is the set of all set-partitionings of L.

If a positive cost C_J is associated with each $S_J \in \mathcal{S}$, the SCP becomes the search for a set-covering of L which has a minimum cost, the cost of $\overline{S} = \{S_{J1},\ldots,S_{Jp}\}$ being $\sum_{i=1}^{P} C_{Ji}$. The Set Partitioning Problem (SPP) is defined correspondingly.

The SCP can be formulated as a (0-1) linear program as follows:

$$\text{Min } Z: \sum_{J=1}^{N} C_J y_J \tag{2.13}$$

$$\text{s.t.} \quad \sum_{J=1}^{N} t_{iJ} y_J \geq 1, \qquad i = 1,2,\ldots,M \tag{2.14}$$

where, y_J and t_{iJ} are as defined in (2.6) and (2.7) respectively.

For SPP the inequalities (2.14) become,

$$\sum_{J=1}^{N} t_{iJ} y_J = 1, \qquad i = 1,\ldots,M \tag{2.15}$$

The T matrix in Fig. 2.1 shows the binary relationship between S_J and ℓ_i.

$$\begin{array}{c|ccccc|}
 & S_1 & S_2 \cdots\cdots & S_J & \cdots\cdots & S_N \\
\hline
1 & & & & & \\
2 & & & & & \\
\cdot & & & & & \\
\cdot & & & & & \\
\ell_i & & & t_{iJ} & & \\
\cdot & & & & & \\
\cdot & & & & & \\
\ell_M & & & & &
\end{array}$$

FIGURE 2.1 - The T Matrix

2.2.2 A Tree Search Algorithm For SPP

The basic difference between SPP and SCP is the existence of no-overcoming restriction in SPP. This fact is very adventageous, while applying a tree search method, since it enables early abandonment of potential branches of the tree.

Christofides suggested to reorder sets before getting in the tree search algorithm. This reordering is called blocking. For each element ℓ_k and L, one block is created. Block k contains these sets which do not cover any of the elements numbered $\ell_1,\ldots,\ell_{k-1}$. Thus, each set S_J can be placed only in one block. For the sake of readibility, these blocks are arranged in tableau format shown in Table 2.1. Depending upon the nature of the problem, some blocks can be empty.

The tree search algorithm moves on the blocks sequentially such that block k is not being searched unless every element ℓ_i, $1 \leq i \leq k-1$, has already been covered in a partial solution.

TABLE 2.1 - The Initial Tableau

ℓ_1	1111	0			
ℓ_2		1111	0		
ℓ_3	0 or 1		1111	0	
ℓ_4		0 or 1		1111	etc.
.			0 or 1	0 or 1	
.					
ℓ_M					

The sets within each block are arranged heuristically in ascending order of their costs. During the course of the tree search besides the sequential search upon blocks, the sets within each block are searched sequentially also. Since the objective is to minimize total cost, search on lower cost sets will be more promising. Then the sets S_J are renumbered such that the set S_J will correspond to the set at the J'th column of the tableau.

While applying the algorithm current best solution $\overline{B}$ and its related cost figure $\overline{Z}$ is kept and updated after every improvement, where $\overline{B}$ is the set of S_J's covered within that best solution. Also B and Z are used to represent the current partial solution at hand, and E shows the elements of L covered by the partial solution B. The steps of the tree search algorithm can be stated as follows:

Initialization:

Step 1 : Perform blocking process to set up the initial tableau and set the partial solution $B = \emptyset$, $E = \emptyset$, $Z = 0$, and let $\overline{Z} = \infty$.

Augmentation:

Step 2 : Find $Q = \min(i | \ell_i \in E)$. Set a marker at the top, i.e. at the lowest cost set of block Q. If block Q is empty, go to step 4; otherwise, continue.

Step 3 : Beginning at the marked position in block Q, examine its sets S_J^Q in increasing order of J.

i) If set S_J^Q is found such that $S_J^Q \cap E = \emptyset$ and $Z + C_J^Q < \overline{Z}$ (where C_J^Q is the cost of S_J^Q) then put marker on set S_J^Q and go to step 5.

ii) Otherwise, if block Q is exhausted or a set S_J^Q is reached for which $Z + C_J^Q \geq \overline{Z}$, then remove last marker and go to step 4.

Backtrack:

Step 4 : B cannot lead to a better solution. If $B = \emptyset$ (i.e. block 1 has been exhausted), terminate with the optimal solution B. Otherwise, remove the last set, S_k^t say, added into B, put $Q = t$, place a marker on set S_{k+1}^t, remove previous marker in block t, set $Z = Z - C$, update E and go to step 3.

If S_{k+1}^t does not exist, then, if $t = 1$, terminate; else go to step 4.

Test for a new solution:

Step 5 : Update $B = B \cup [S_J^Q]$, $E = E \cup S_J^Q$, $Z = Z + C_J^Q$. Remove last marker. If $E = L$ a better solution has been found. Set $\overline{B} = B$, $\overline{Z} = Z$ and go to step 4. Otherwise, go to step 2.

Since the search terminates with the exhaustion of block 1 at step 4, it would be better to arrange blocks in ascending order according to the number of sets in each block. This can be achieved by renumbering the elements $\ell_1, \ldots, \ell_M$ in increasing order of number of sets in S containing that element, before setting up the initial tableau.

Christofides has suggested some dominance tests which will improve algorithm. Some of these can be stated in short as follows:

Keep for each value of $Z = 1, 2, \ldots, \overline{Z}$. Some (perhaps incomplete) list of maximal E's which have been achieved for this Z, (where by maximal is meant a set not included in another set which is also in the list). These lists E's can then be used to limit the search by eliminating branches that later on prove fruitless.

Chrisdofides has also suggested some methods of findings a lower bound on the cost of the branches obtained during the course of the algorithm. But we shall not make use of these bounds throughout this study, so we have omitted that part.

There are also some other methods proposed for solving SPP. Pierce and Lasky (1973) give some modifications to the above basic algorithm, including subsidiary use of a linear program. Michadu (1972) describes another implicit enumeration algorithm which is based on a linear programming problem corresponding to SPP with the block structure given above being used in a secondary role.

Other algorithms involving simplex-type iterations have been proposed, both primal (Balas and Padberg, 1972) and dual (Jensen, 1971, and Salkin and Kencal, 1970).

Defining some derivatives of his problem will be of more interest to us and thus we shall define some new problems.

2.3 THE BOTTLENECK SET PARTITIONING PROBLEM (BSPP)

BSPP differs from classical SPP only in the form of the objective function. SPP formulation tries to minimize the total cost of sets within $\overline{S}$. But BSPP formulation tries to minimize the maximum cost within $\overline{S}$. Thus, the formulation in Section 2.2.1 becomes,

$$\text{Min } Z = \min_{\overline{S}_\ell \varepsilon \overline{\mathcal{S}}} [\max_{S_J \varepsilon \overline{S}_\ell} [C_J]] \tag{2.16}$$

$$\text{s.t.} \sum_{J=1}^{N} t_{iJ} y_J = 1 \qquad i = 1,2,\ldots,M \tag{2.17}$$

where y_J, t_{iJ} and C_J are as defined in the SPP formulation in Section 2.2.1.

2.4. THE RESOURCE CONSTRAINED SET PARTITIONING PROBLEM (RCSPP)

RCSPP is essentially a generalized version of SPP, RCSPP formulation is built upon the SPP formulation by introducing a set of resource balance constraints.

Resource constraints are imposed as follows. Assume that each set $S_J \varepsilon \overline{S}$ consumes exactly one unit of some available resource. Let there be several resource types and each set can consume only one type of resource. That is,

$$\sum_{k=1}^{RP} h_{Jk} = 1 \qquad \text{for} \quad J = 1,\ldots,N \tag{2.18}$$

where

$$h_{Jk} = \begin{cases} 1, & \text{if set J consumes resource k} \\ 0, & \text{otherwise} \end{cases} \tag{2.19}$$

and

RP: is the number of resource types.

Then, we can formulate RCSPP as follows:

$$\text{Min } Z = \sum_{J=1}^{N} C_J y_J \tag{2.20}$$

$$\text{s.t.} \sum_{J=1}^{N} t_{iJ} y_J = 1 \quad , \qquad i = 1,2,\ldots,M \tag{2.21}$$

$$\sum_{J=1}^{N} h_{Jk} y_J = H_k, \qquad k = 1,2,\ldots,RP \tag{2.22}$$

where H_k: is the available amount of resource type k, and C_J, y_J, t_{iJ} defined as earlier.

Depending on the value of RP, we have two cases:

i. When RP = 1, it means that there exists only one type of resource. In this case, RCSPP formulation forces the SPP formulation so that the solution vector $\overline{S}'$ has exactly H_1 components.

ii. When RP > 1, then the solution vector $\overline{S}'$ is forced to have exactly $\sum_{k=1}^{RP} H_k$ components, and resource consumption of the components are forced to levels given in the resource availability vector H.

Of course, there may be cases where the resource consumption equation is relaxed and equality sign in the resource availability constraint is replaced by a less than or equal sign. Then, the solution vector is forced to have components less than or equal to the prespecified levels.

2.5. THE BOTTLENECK RESOURCE CONSTRAINED SET PARTITIONING PROBLEM (BRCSPP)

BRCSPP combines the characteristics of both BSPP and RCSPP. That is, it both forces the solution vector have a prespecified amount of components and tries to minimize the maximum cost of the sets within the solution vector instead of minimizing the total cost of the set-covering.

Thus, the combined formulation becomes,

$$\text{Min } Z = \min_{\bar{S}_\ell \varepsilon \bar{\mathcal{S}}} [\max_{S_J \varepsilon \bar{S}_\ell} [C_J]] \tag{2.23}$$

$$\text{s.t.} \sum_{J=1}^{N} t_{iJ} y_J = 1 , \qquad i = 1,2,\ldots,M \tag{2.24}$$

$$\sum_{J=1}^{N} h_{Jk} y_J = H_k , \qquad k = 1,2,\ldots,RP \tag{2.25}$$

where t_{iJ}, y_J, h_{Jk} are as defined earlier.

The formulation of BRCSPP is exactly equivalent to the mathematical programming formulation of BRP, hence if we solve BRCSPP, we get the solution to BRP. Also, the formulation of RCSPP is exactly equivalent to the mathematical programming of MCRP, hence the same statement holds for that case.

2.6. ULUSOY'S ALGORITHM FOR BRP

For the solution of BRP Ulusoy (1981) has suggested to modify the SPP algorithm given by Christofides (1975) so that to handle both the minimax objective function and the resource constraints.

The improvements can be summarized as follows:

i. *The modifications to handle resource constraints.*

As in the classical SPP, the current partial solution is kept by three variables, B, Z, and E. But in addition to these a new vector G is introduced which keeps track of the current usage of resources, i.e. G_k is the current amount of resource to be used. When a new set is added to the partial solution B, the G vector is updated, i.e. the current usage of resource type of lately introduced set is increased by one. Similarly when a set is drawn out of the partial solution, then the corresponding component of G is decreased. If the current usage of resource type k is equal to the availability of that resource then the sets consuming resource k are disregarded at step 3(i) of the SPP algorithm, i.e. first the resource availability is checked. If all available resources have been used up, then the sets consuming this resource is disregarded at step 3(i).

ii. The modifications for minimax objective function.

The RCSPP problem is solved by the modified SPP algorithm with given sets. Then, among the feasible solutions generated during this solution process, the one which minimizes the maximum set cost is chosen without considering the minimum total objective function value, i.e.,

$$Z^* = \min_{\overline{S}_\ell \varepsilon \mathcal{S}}[\max_{S_J \varepsilon \overline{S}_\ell} [C_J]]$$

Then, the sets in $\mathcal{S}$ are scanned and the ones whose cost is greater than or equal to Z^* are deleted. Thus $\mathcal{S}$ is reduced and the routine is reinitiated on remaining sets. This process is repeated until there is no feasible solution on remaining sets with the last solution being an optimal solution to BRP.

Ulusoy (1983) has further improved this approach in two aspects:

i. An upper bound on the length of mission time is provided by a heuristic procedure. The heuristic procedure produces a good feasible solution to BRP which results in relatively reduced number of paths processed by the specialized set partitioning routine.

ii. The first version of the procedure required the execution of the set partitioning routine several times and one final complete enumeration. In this version, a new labelling routine for the paths has been introduced which produces the exact optimum by only a single scan through the first block, thus resulting in a large computational saving.

A further improvement can be introduced by the following suggestion.

Let us suppose, there is only one type of resource. Then as it was mentioned earlier the SPP algorithm is forced to get a given number of components into the solution set.

Now, specifically assume there are 20 rows (M = 20) and only 4 resources (RP = 4) available. Then the problem becomes: choose 4 or less sets so as to cover 20 rows. Now assume a partial solution B, Z, and E which uses 3 sets, thus consumes 3 resources and covers 10 rows. In this situation, SPP routine looks to the uncovered rows, chooses the minimum index of uncovered rows at step 2 and enters that block and searches for a feasible solution at step 3. But at this point one can decide whether a feasible solution may exist or not.

One can look at the remaining sets. If there is no set S_J, where the number of rows covered by the set S_J is greater than or equal to 10, then, one can conclude that this branch is fruitless. In that case, there is no possibility of reaching a feasible solution. Since one can use only one more set, but there is no set which is covering at least 10 rows. If there exists such a set, then there is a possibility of reaching a feasible solution.

Although this is an exaggarated example, most of the time similar cases decrease the efficiency of the SPP routine.

If there are more than one resource type, then the difficulty again arises due to the same reasoning.

III. THE SOLUTION PROCEDURE DEVELOPED FOR THE "BOTTLENECK ROUTING PROBLEM"

3.1 INTRODUCTION

With the intent of improving upon the solution procedure based on specialized set partitioning algorithm a new method is developed here. The method is designed to find an exact optimum.

The method suggested approaches to the problem just in the opposite direction as compared to the set-partitioning approach. Instead of eliminating the sets from the set ϕ gradually, this technique starts with no set at hand and gradually enlarges ϕ.

Before getting in the details of this new method we shall first concentrate on the calculation of S_J and C_J values defined in Chapter 2.

3.2 THE NETWORK TRANSFORMATIONS REQUIRED TO OBTAIN S_J and C_J

The original network of the BRP is a fully connected symmetric network, where the nodes of this network represent the airports. This network is symmetric since the flight time from airport i to airport J is practically equal to the flight time from airport J to airport i. But symmetry is assumed only for the sake of simplicity. The solution technique is also applicable to the assymmetric cases (See Example E). Let D be the flight time matrix, where d_{iJ} is the flight time from airport i to airport J.

From this network a transformed network is formulated in order to generate the sets (S_J's) and related cost figures (C_J's), defined in Section 2.1.

In this transformed network, each S_J will be defined by a unique simple path, and the length of that path will correspond to the related C_J value. Practically C_J indicates the time required to traverse that path (that is the time required to carry all the loads on the related path) and will be called as "path length" from now on.

Originally this network transformation is suggested by Ulusoy (1981) and his suggestion will be introduced in Section 3.2.4.

The transformed network is obtained in two phases. In the first phase nodes, and in the second phase arcs of the transformed network are generated.

3.2.1 Node Transformations

In the transformed network, there is one node for each of the airports, which has planes initially, in the original network, and there is one node for each of the loads that should be carried, and there are two artificial nodes. Thus, totally there are,

$$Y = RP + M + 2 \tag{3.1}$$

nodes. Namely,

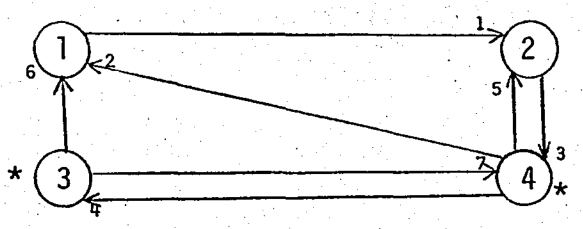
Node 1: refers to the artificial source node.

Nodes 2, ..., RP+1: correspond to the airports which have planes initially.

Nodes RP 2,...,RP+M+1: correspond to the loads 1,2,...,M respectively.

Node RP+M+2: refers to the artificial terminal node.

In the Example A, there are 4 airports and 7 loads which should be transported between these airports. These loads are indicated by arcs in the original network (See Fig. 3.1). Original locations of the planes are assumed to be the airports 3 and 4.



Note: Nodes correspond to airports.

* indicates that planes are available at these airports initially.

The arcs between the airports show the loads that are to be carried between these airports.

FIGURE 3.1 - Original network of Example A

$$D = \begin{array}{c|cccc|} 1 & - & 30 & 65 & 40 \\ 2 & 30 & - & 65 & 35 \\ 3 & 65 & 30 & - & 20 \\ 4 & 40 & 35 & 20 & - \end{array}$$

Note: times taken in minutes.

TABLE 3.1 - Flight Time Data of Example A

From this original network, the transformed network is obtained with 11 nodes, as shown in detail in Fig. 3.2. In this network nodes represent the following.

Node 1:	The artificial source node.
Node 2:	Airport 3.
Node 3:	Airport 4.
Node 4:	Load 1 (from airport 1 to airport 2)
Node 5:	Load 2 (from airport 4 to airport 1)
Node 6:	Load 3 (from airport 2 to airport 4)
Node 7:	Load 4 (from airport 4 to airport 3)
Node 8:	Load 5 (from airport 4 to airport 2)
Node 9:	Load 6 (from airport 3 to airport 1)
Node 10:	Load 7 (from airport 3 to airport 4)
Node 11:	The artificial terminal node.

3.2.2 Arc Transformations

The transformed network is assymmetric and is not fully connected. Let the time matrix related with this network be W. This matrix is Y by Y and called the "operation time matrix".

It would be better to consider the arc transformations in far stages.

3.2.2.1 Stage 1: Interactions Between the Loads

In this stage, the interactions between nodes representing loads in the transformed network are considered. That is, nodes RP+2,...,RP+M+1 are considered. All these nodes are fully connected to each other and the operation times ($\omega_{k\ell}$) between these nodes are calculated as follows:

Case a: If the ending airport of the initial node (load in this stage) coincides with the starting airport of the final node in the transformed network, then the time required to traverse this arc (from initial node to final node) is, the flight time from starting airport to ending airport of final node, plus the loading time (TL) plus the unloading time (TU) (See Arc A on Fig. 3.2). That is,

$$\omega_{k\ell} = d_{[L(\ell-1-RP),1],[L(\ell-1-RP),2]} + TU + TL \tag{3.2}$$

$$\forall k,\ell = RP+2,\ldots,RP+M+1 \,, \qquad k \neq \ell$$

$$\text{and if } L_{(k-1-RP),2} = L_{(\ell-1-RP),1}$$

where, $L_{i,1}$: is the starting airport of load i.

$L_{i,2}$: is the ending airport of load i.

Thus, $L_{(\ell-1-RP),1}$ is the airport corresponding to starting airport of (ℓ-1-RP)'th load, where the (ℓ-1-RP)'th load corresponds to the ℓ'th node by the definition in Section 3.2.1.

Case b: If the ending airport of the initial node does not coincide with the starting airport of the final node, then this means there exists an empty flight (a flight in which the associated plane does not carry any load) between these nodes, and the time required for that flight should be added to the time required to traverse that arc (See Arc B on Fig. 3.2). That is,

$$\omega_{k\ell} = d_{[L_{(k-1-RP),2}],[L_{(\ell-1-RP),1}]}$$

$$+ d_{[L_{(\ell-1-RP),1}],[L_{(\ell-1-RP),2}]} + TU + TL \qquad (3.3)$$

$\forall\ k,\ell = RP+2,\ldots,RP+M+1, \quad k \neq \ell$

and if $\quad L_{(k-1-RP),2} \neq L_{(\ell-1-RP),1}$

3.2.2.2 Stage 2: Interactions Between the Airports Which Have Planes Initially and the Loads

In this stage the interactions between the nodes representing airports which have planes initially and the nodes representing loads are considered. There are arcs from all nodes representing airports to all nodes representing loads, but there are no arcs in the counter direction. The operation times are calculated as follows:

Case a: If the airport at the initial node coincides with the starting airport of the final node, then a case similar to Case a of Stage 1 occurs (See Arc C on Fig. 3.2). That is,

$$\omega_{k\ell} = d_{[L_{(\ell-1-RP),1}],[L_{(\ell-1-RP),2}]} + TU + TL \qquad (3.4)$$

for $k = 2,\ldots,RP$

$\ell = RP+2,\ldots,RP+M+1$

and if $L_{(\ell-1-RP),1} = (k-1)^{th}$ airport which has planes initially.

Case b: If the airport at the initial node does not coincide with the starting airport of the final node, then a case similar to Case b of Stage 1 occurs (See Arc D on Fig. 3.2). That is,

$$\begin{aligned}\omega_{k\ell} &= d_{[(k-1)^{th}\text{ airport which has planes initially}],[L_{(\ell-1-RP),1}]} \\ &+ d_{[L_{(\ell-1-RP),1}],[L_{(\ell-1-RP),2}]} + TU + TL\end{aligned} \tag{3.5}$$

for $k = 2,\ldots,RP$

$\ell = RP+2,\ldots,RP+M+1$

and if $L_{(\ -1-RP),1} \neq (k-1)^{th}$ airport which has planes initially

3.2.2.3 Stage 3: Interactions Between the Artificial Source Node And the Airports Which Have Planes Initially

The artificial source node and the artificial terminal node is introduced so that to generate a complete network for the reasons discussed in the next sections.

All arcs between artificial source node and the nodes representing airports which have initially planes are dummy and their operation times are zero. There are no arcs in opposite direction (See Arc E on Fig. 3.2). That is,

$$\omega_{1\ell} = 0 \quad , \quad \text{for} \quad \ell = 2,\ldots,RP+1 \tag{3.6}$$

3.2.2.4 Stage 4: Interactions Between the Loads and the Artificial Terminal Node

Again, all arcs between the nodes representing loads and the terminal node are dummy and have zero operation time and no arcs in opposite direction (See Arc F on Fig. 3.2). That is,

$$\omega_{k,RP+M+2} = 0 \quad , \qquad k = RP+2,\ldots,RP+M+1 \tag{3.7}$$

3.2.3 An Example of Network Transformations

The node transformations of Example A is given in Section 3.2.1. The flight time matrix D given in Table 3.1, and the loading and unloading times are taken as 10 and 5 minutes respectively. The load numbers are indicated on Fig. 3.1.

Following example computations given to illustrate the interactions described in previous sections, and the complete operation time matrix is given in Table 3.2.

a) Stage 1: Case a

$$\omega_{4,6} = d_{[L_{(6-1-2),1}],[L_{(6-1-2),2}]} + TU + TL$$

$$= d_{2,4} + TU + TL$$

$$= 50 \text{ minutes}$$

since $L_{1,2} = L_{3,1}$

This computation is related with Arc A on Fig. 3.2.

b) *Stage 1: Case b*

$$\omega_{4,7} = d_{[L_{(4-1-2),2}],[L_{(7-1-2),1}]}$$

$$+ d_{[L_{(7-1-2),1}],[L_{(7-1-2),1}]} + TU + TL$$

$$= 70 \text{ minutes}$$

c) *Stage 2: Case a*

$$\omega_{2,9} = d_{[L_{(9-1-2),1}],[L_{(9-1-2),2}]} + TU + TL$$

$$= 80 \text{ minutes}$$

Since $L_{6,1} = 3$

d) *Stage 2: Case b*

$$\omega_{3,10} = d_{4,[L_{(10-1-2),1}]} + d_{[L_{(10-1-2),1}],[L_{(10-1-2),2}]}$$

$$+ TU + TL$$

$$= 55 \text{ minutes}$$

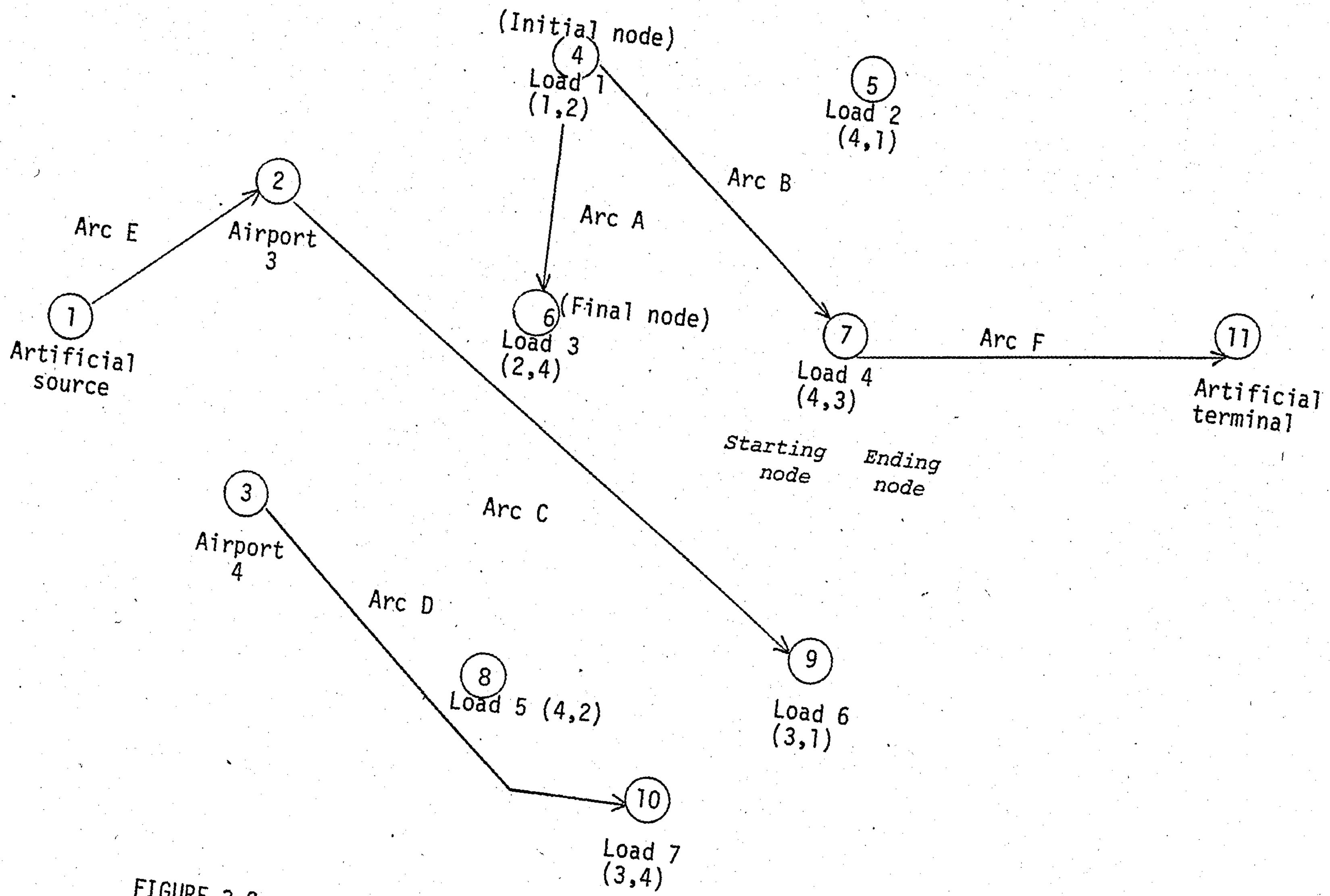


FIGURE 3.2 - Transformed network of Example

TABLE 3.2 - The Operation Time Matrix Obtained From the Transformed Network of Example A

$$W = \begin{array}{c|ccccccccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
1 & \infty & 0 & 0 & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\
2 & \infty & \infty & \infty & 110 & 75 & 80 & 55 & 70 & 80 & 35 & \infty \\
3 & \infty & \infty & \infty & 85 & 55 & 85 & 35 & 50 & 100 & 55 & \infty \\
4 & \infty & \infty & \infty & \infty & 90 & 50 & 70 & 85 & 110 & 65 & 0 \\
5 & \infty & \infty & \infty & 45 & \infty & 80 & 75 & 90 & 145 & 100 & 0 \\
6 & \infty & \infty & \infty & 85 & 55 & \infty & 35 & 50 & 100 & 55 & 0 \\
7 & \infty & \infty & \infty & 110 & 75 & 80 & \infty & 70 & 80 & 35 & 0 \\
8 & \infty & \infty & \infty & 75 & 90 & 50 & 70 & \infty & 110 & 65 & 0 \\
9 & \infty & \infty & \infty & 45 & 95 & 80 & 75 & \infty & \infty & 100 & 0 \\
10 & \infty & \infty & \infty & 85 & 55 & 85 & 35 & 50 & 100 & \infty & 0 \\
11 & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0
\end{array}$$

3.2.4 The Network Transformation Suggested by Ulusoy

Ulusoy (1981) has formulated the transformed network in a slightly different way.

According to his formulation there are (R+M) nodes. That is, all airports are placed in this network without caring whether they have planes initially or not. He also put one node for each of the loads as in our case.

He omitted the interactions discussed in Stage 1 case b and Stage 2 case b. That is, the cases which enable empty flights between airports are omitted. In order to represent empty flights he has defined two set of new interactions.

i) To enable empty flights at the first leg, all nodes representing airports are interconnected with arcs whose operation times equal to the flight times which they represent.

ii) To enable empty flights later in the routes, all nodes representing loads are connected to all nodes representing airports, except to the airport same as the ending airport for that load and the associated flight times are assigned as the operation times.

3.3 Computation of S_J and C_J

Using the transformed network and the operation time matrix W, we shall obtain S_J and C_J by enumerating all "simple paths" from the artificial source node to artificial terminal node. By a simple path between any two nodes of a network, we mean a path with no repeating nodes. Such an enumaration will lead to paths like:

$$[S,p,\ell_1,\ell_2,\ldots,\ell_k,T] \tag{3.8}$$

where, S and T represent artificial source and terminal nodes respectively.

Also,

$p \varepsilon \{2,\ldots,RP+1\}$ i.e., p refers to one of the airports which has planes initially

and $\ell_i \varepsilon \{RP+2,\ldots,RP+M+1\}$ i.e., ℓ_i's refer to the loads covered by that path.

This sequence of nodes is guaranteed for all the simple paths since there are no arcs directed from nodes representing loads to nodes representing airports and to source node. Also there are no arcs directed from nodes representing airports to terminal and source nodes. Node T is forced to be the terminal node since there are no arcs leaving T, just as node S is forced to be the source node.

Therefore, by enumerating all the simple paths between nodes S and T over this transformed network, we can obtain all possible combinations for S_J. That is, all possible combinations of loading a plane with different groups of loads can be obtained. We can differentiate between initial airports, since these simple paths involve initial airports. By this way, these paths will directly give us all S_J and h_{Jk} combinations and their cost figures (C_J's) being the time required to traverse that path. That is, length of that path over the matrix W.

In general enumerating all the simple paths in a dense network is practically impossible due to the tramendous number of combinations. But in the case of the bottleneck routing we do not require all the combinations. Only the ones which are shorter than a prespecified length, d_{max}, are needed.

There are two ways of establishing value of d_{max}.

i) d_{max} can be decided upon a priori as a result of operational requirements and be given.

ii) If no such a priori decision exists, then the analyst can estimate the latest mission completion time by simply overviewing the situation.

3.4 ENUMERATION OF SIMPLE PATHS SHORTER THAN A GIVEN LENGTH

For enumerating simple paths in a graph many techniques have been developed. Solving this problem is often the first step of important procedures like symbolic network analysis or terminal reliability computations in a communication network (Fratta and Montari, 1975).

Two essentially different techniques can be extracted from the wide literature on this problem; the routing technique and the matrix technique.

The routing technique (Lin and Anderson, 1969) and Kroft, 1967) is useful mainly for enumerating all paths between a single pair of nodes.

The matrix technique (Danielson, 1968) is based on computing symbolic powers of the graph adjacency matrix. In fact, each element (i,J) of the m'th power of the adjacency matrix contains all paths of length m between nodes i and J. This technique reduces the combinatorial explosion since it erases nonsimple paths during the execution of the procedure.

Fratta and Montanari (1975) introduced a path algebra and translated the path problem into a system of linear equations in this algebra, which they solved by an iterative method.

In order to enumerate the simple paths in the transformed network, I have chosen the routing technique for the following reasons:

i) Although it is an exhaustive search procedure, it requires very little memory. Only the adjacency matrix needs to be stored besides some negligible control arrays.

ii) It is essentially devoted to the problem of enumerating paths between given nodes. So, unnecessary effort, such as trying to generate the paths between all nodes is avoided.

iii) The procedure seperately traces up all nonsimple paths up to the second occurrence of the first repeated node. That is, forms the path by gradually adding new nodes. This fact is advantageous while generating paths shorter than a prespecified length.

Lin and Anderson (1969) have given major steps of the algorithm as follows:

Step 1: Define a fixed ordering of the arcs starting from each node. Let A be the first arc starting from S. Mark S and T.

```
Step 2: Let $V_2$ be the node where A terminates at if $V_2$ is unmarked,
        then, if no arcs start from $V_2$,
              then, go to step 3
              else, mark $V_2$ and A
                    rename A with first arc starting from $V_2$
                    go to step 2
        else, if $V_2 = T$,
              then, a simple path between S and T is obtained.
                    store it
                    go to step 3.
              else, go to step 3.

Step 3: Let $V_1$ be the node where A starts from. if A is the last
        arc starting from $V_1$,
        then, if $V_1 = S$
              then, stop all paths are generated.
              else, erase mark from $V_1$ and A.
                    rename A with the marked arc terminating at $V_1$.
                    go to step 3.
              else, rename A with the successor of A in the ordering
                    relative to $V_1$, go to step 2.
```

This algorithm is revised to generate S_J's and C_J's more efficiently. But before describing these, we must mention another simple transformation. As it was defined, arcs leaving the nodes S and arcs entering the node T have zero time. At this stage a positive time ε assigned to such arcs, since we identify an arc as marked or unmarked by looking at its sign in the operation time matrix W. Thus, due to the structure of the transformed network all the path lengths will be increased by 2ε. Therefore d_{max} is updated as (d_{max} 2ε) before starting the procedure. Also, note when recording the paths their lengths should be decreased by 2ε.

The routing algorithm suggests the use of adjacency matrix. But by the use of weighted adjacency matrix, we can still identify arcs, furthermore store their lengths. Also keep marks of these arcs on this matrix, by keeping track of their signs. Therefore, we only need an array to control mark of nodes. We do not need to write down all the arcs starting from each node if we move sequentially on the weighted adjacency matrix.

Flowchart 1 illustrates the steps of the revised form of the routing technique.

3.5 PATH ELIMINATION

Due to the nature of path generation algorithms they generate all permutations of paths between given nodes. As an example, assume in Example A if path 1,2,9,8,11 will also be generated unless their lengths are greater than d_{max}. But according to our definition (See Eq. (2.9)) the shorter one should be kept and rest should be eliminated.

Although the number of paths does not bring any problem during generation phase, the elimination process creates necessity of keeping the paths in the memory. It is also possible to make elimination without storing the paths in the memory, but it will be very time consuming. Therefore, if the number of paths are on the order of few thousands, then it would be faster to perform this task by storing paths in the computer memory.

In order to speed up this process following ideas have been developed. Assume that the paths obtained after the execution of path generation routine stored in one of the mass storage devices of the computer system used.

The major problem elimination process is to identify whether any two paths cover the same nodes or not. In order to make this identification easier, every node is assigned a random odd integer, as a dummy demand. Then, for each path, the sum of node demands on that path will give the demand of that path. Thus, if any two paths cover the same nodes, then their demands are equal.

We can summarize the elimination process as follows. Note, $\overline{D}_d$ shows the first occurrence sequence number of a path with demand d, in the memory.

<u>Step 1</u>: Set $i = 0$ and $\overline{D}_d = 0$ for all possible d values.

<u>Step 2:</u> Read next path from mass storage. Calculate its demand d. If $\overline{D}_d = 0$, then, go to step 4; otherwise, go to step 3.

<u>Step 3</u>: Let $J = \overline{D}_d$. Starting from J'th position in the memory check whether there exists any path covering the same nodes, with last read one. If such a path found, then, select the shorter, and locate to that position; otherwise, go to step 4.

Step 4: Set i = i+1. Record the last read path, to the memory as i'th path. Set $\overline{D}_d$ = i. Go to step 2.

In Flowchart 2 the detailed steps of this process is illustrated.

3.6 SORTING THE PATHS

Algorithms used in this thesis for the bottleneck routing of military cargo airplanes require the paths to be sorted according to their lengths in an ascending order. Therefore after the elimination process the paths have to be sorted.

For this purpose there are a variety of methods reported in literature (Knuth, 1975). Depending upon the number of paths generated two methods have been used in this thesis.

If there is a reasonable number of paths, then, the well-established Heap-sort technique is used and all the paths are sorted in one step.

If the number of paths is large, then, one should resort to more sophisticated sorting techniques. Most common technique is to divide data into reasonable sized groups. Then, sort each group independently and merge these sorted groups. Since the UNIVAC 1106 Operating System has a SORT-MERGE package currently available no program is developed for sorting large amounts of paths. But the program listings of the first technique can be seen on Appendix F.

After the gnereation, elimination and sort operations, the paths are stored on a sequential access data file on one of mass storage devices of the computer, which will be called as "mass storage" in short from now on.

3.7 THE ALGORITHM DEVELOPED FOR BOTTLENECK ROUTING PROBLEM

3.7.1 Introduction to BRP Algorithm

As it was explained at the beginning of this chapter, this algorithm approaches the BRP in a different fashion. That is, we start with no path at hand and enlarge S gradually. The set of paths that are currently available at hand at any stage of the algorithm is called the "path list". Also, the term "load cardinality" is used to express number of loads covered by the associated path.

We can briefly summarize the algorithm as follows,

Step 1: Initialize

Get some paths from mass storage into the path list until some conditions are satisfied.

Step 2: Search

Search over the path list. If there exists a solution then stop. The solution is an optimal solution to the BRP. Otherwise continue.

Step 3: Enlarge

Enlarge the path list by getting some more paths from mass storage. Go to step 2.

The following sections will explain each step in detail while giving the necessary proofs.

3.7.2 The First Step: Initialize

At the beginning of the Initialization step, we have no paths at hand and we begin by taking paths into the path list. We sequentially take the paths from mass storage until a change in path length is observed. When such a change occurs, we shall apply the following rules in order to detect whether the paths at hand can give rise to a feasible solution or not. If so, we shall go the Search step to locate that solution. Otherwise we shall continue taking paths from mass storage until another length change occurs.

The Stopping Rules:

Rule 1: Coverage Check

If $f_{11} < M$, then there cannot be solution in the given path list, where

$$f_{iJ} = \begin{cases} \text{number of times node J occurred in a path with load cardinality i, within given path list; if } J > 1. \\ \text{number of disjoint loads on paths with load cardinality i; if } J = 1 \end{cases} \qquad (3.9)$$

If the number of disjoint loads covered by the paths whose load cardinality is one is less than the total number of loads, then there cannot be a solution. It is enough to check the paths whose load cardinality is one in order to detect whether all loads covered or not by the given path list, due to the following proposition.

Proposition 3.1: The first occurrence of every load will be in a path whose load cardinality is one, if the paths are ordered by the process defined in Sections 3.1 through 3.6.

Proof: The proof follows from triangular inequality which holds, since the Euclidean metric is valid here.

Q.E.D.

The statement of Proposition 3.1 can be generalized as follows,

Corollary 3.1: Any load $\ell \varepsilon \{1,\ldots,M\}$ cannot appear in a path whose load cardinality is $(k+1)$ before appearing in a path whose load cardinality is k, for $k = 1,2,\ldots$

Before introducing Rule 2, 3, and 4, let us first make the following definitions.

Definition 3.1: The configuration vector is the vector which shows the number of loads assigned to each plane. Let,

$$Q = \left| \frac{M}{P} \right| \qquad (3.10)$$

where the operation stands for integer division and let K be the remainder of this integer division.

<u>Definition 3.2</u>: The configuration vector, in which the number of paths whose load cardinalities are large is minimum, is called <u>the minimum configuration</u>.

If $G_{max} = Q+1$, where G_{max} is the largest load cardinality in the given path list, then the vector,

$$\overline{C} = \underbrace{Q,Q,\ldots,Q}_{P-K},\ \underbrace{Q+1,\ldots,Q+1}_{K}, \qquad K > 0 \tag{3.11}$$

is the "minimum configuration" vector indicating that K planes carry (Q 1) loads and (P-K) planes carry Q loads.

If $K = 0$, then the minimum configuration occurs when $G_{max} = Q$ and indicates that P planes carry Q loads.

Total number of loads handled by minimum configuration is M, i.e., all loads are carried by the minimum configuration. This can be easily shown as follows,

$$\begin{aligned}(P-K)Q + K(Q+1) &= PQ - KQ + KP + K \\ &= PQ + K \\ &= M\end{aligned}$$

when $G_{max} = Q+1$, there are other configurations such as,

$$C = [\underbrace{Q,\ldots,Q}_{P-K-1},\ \underbrace{Q-1}_{1},\ \underbrace{Q+1,\ldots,Q+1}_{K+1}] \tag{3.12}$$

But such configurations necessitates more loads to be covered by paths of load cardinality (Q+1) than the minimum configuration.

As an example, assume that there are 14 loads and 4 planes, (i.e., $M = 14$ and $P = 4$). This results in $Q = 3$, $K = 2$, and $\overline{C} = [3,3,4,4]$. Assume that $G_{max} = 4$, then we have to construct a solution by utilizing paths of load cardinalities of at most 4, and $\overline{C}$ is that configuration of solutions which minimizes the use of paths of load cardinalities 4. For this case, the configuration defined in Eq. (3.12) will be $\overline{C} = [2,4,4,4]$. This configuration utilizes more paths of load cardinality 4 than minimum configuration, which implies that longer paths must be utilized and thus more paths must be processed in the path list.

Thus, if we resort to full enumeration, where we generate paths of load cardinality K only after we have generated all paths of load cardinality (K-1), then we shall first catch a solution with minimum configuration.

Rule 2: Member Size Check

Let,

$$M1 = \begin{cases} Q + 1 & , \text{ if } \quad K \neq 0 \\ Q & , \text{ if } \quad K = 0 \end{cases} \tag{3.13}$$

If $G_{max} < M1$, then there cannot be a solution generated from the available path list. Since, in that case total loads carried by P planes can never sum up to the total number of loads as can easily be shown.

In the previous example, if there is no path with load cardinality greater than 3, then there cannot be a solution with 4 planes, since the maximum number of loads that can be carried would be 12, whereas there are 14 loads to be carried.

Rule 3: Minimum Configuration Check

If $G_{max} = M1$ and $f_{M1,1} < (M1 \times K)$, then, there cannot be a solution generated from the available path list.

This rule is more strict form of Rule 2. In this case, we consider the number of disjoint loads covered by the paths of load cardinality M1. Since no path with load cardinality (Q+2) have been read from mass storage ($G_{max} = M1$), the condition for minimum configuration is satisfied and we need at least (M1 x K) loads covered by paths of load cardinality M1.

To illustrate this rule, consider the previous example again. We require at least 8 loads to be covered by paths of load cardinality 4, if no path with load cardinality 5 have yet occurred. This means, we need at least 2 planes be allocated to paths of load cardinality 4, with disjoint loads.

Definition 3.3: The Worst Load

Let

$$G_{min} = \underset{\text{loads } \ell}{\overset{\text{min}}{\text{overall}}} \left\{ \begin{array}{l} \text{maximum load cardinality} \\ \text{of the paths in which} \\ \text{load } \ell \text{ has occurred so far} \end{array} \right\} \quad (3.14)$$

Then,

W = is the load ℓ for which the minimum of expression (3.16) has occurred is called the Worst Load.

<u>Rule 4</u>: Worst Load Check

Let,

$$REM = Q - G_{min} \tag{3.15}$$

$$\overline{M1} = M1 * (K+REM) \tag{3.16}$$

If $G_{max} = M1$ and $f_{M1,1} < \overline{M1}$, then, there cannot be a feasible solution generated from the available path list.

Note that, this rule does not apply when REM + 0.

Since $G_{max} = M1$, there are no paths in the path list whose load cardinality is (Q+2). Therefore, paths with load cardinality M1, should cover more loads in this case. Because at least one load should be covered by a path whose load cardinality is less than Q (i.e., load W should be covered in a path whose load cardinality is G_{min}).

This rule says paths with load cardinality M1 should cover $\overline{M1}$ loads.

$$\begin{aligned}\overline{M1} &= M1 \times (K+REM) \\ &= M1 \times (K+Q-G_{min}) \\ &= M1 \times K + M1(Q - G_{min})\end{aligned}$$

M1 loads should be covered by Rule 3 and $(Q - G_{min})$ is the gap brought by the situation of load W and this gap should be covered by paths of load cardinality (Q+1).

As an example, assume that there are 32 loads and 6 planes (i.e., M = 32 and P = 6). This results Q = 5, K = 2, $\overline{C}$ = [5,5,5,5,6,6] and M1 = 6. Also assume G_{max} is 6 and largest cardinality of paths covering one of the loads is 3. Thus, $G_{min} = 3$; REM = 2 and $\overline{M}1 = 6(2+2) = 24$. If there exists a solution, then that solution should cover one path of load cardinality 3 and the configuration will be 5,3,6,6,6,6 , which implies that at least 24 loads should be covered by paths of load cardinality 6.

Rule 5: Minimum Plane Check

This rule applies no matter what the value of G_{max} is. A lower bound on the number of planes is established by checking the path list at hand. If this lower bound is less than the plane availability, then, there cannot be any solution generated from the path list at hand.

Essentially planes are assigned to paths without checking any of the BRP constraints and these assignments are made on paths whose load cardinalities are larger (as far as possible).

It is better to explain the principle of the minimum plane check on an example.

Let us suppose that we have 14 loads to cover and the given path list has the following characteristics:

6 loads are covered by paths of load cardinality 4

4 loads are covered by paths of load cardinality 3

3 loads are covered by paths of load cardinality 2

1 loads are covered by paths of load cardinality 1

Since the following analysis is for the minimum number of planes necessary, let us assume that load and path combinations are such that all the assignments indicated, are possible.

Start by assigning one of the planes to a path with load cardinality 4. So 4 loads are covered. No two planes can be assigned to paths of load cardinality 4 since this will result to an infeasible solution to BRP anyhow. The remaining 2 of the 6 loads that should be covered paths whose load cardinality 4, in this case will be covered by paths whose load cardinality 3. By Corollary (3.1) to Proposition (3.1), there have to be paths with load cardinality 3 covering these loads. Thus with the additional 2 loads, 6 of the loads are to be covered by paths with load cardinality 3, now. So, we make 2 plane assignments to paths with load cardinality 3, which makes a total of 10 loads covered by 3 planes. Similarly, one plane is assigned to path with load cardinality 2 and the remaining 2 loads are then covered by paths of load cardinality 1. Thus, a total of 6 planes are required. If the plane availability is less than 6, then, there cannot be a solution within this path list.

We can summarize the process as follows,

<u>Step 1:</u> Let $\bar{f}_J$ = the number of loads covered by paths of load cardinality J, $(J = 1,\ldots,G_{max})$. (3.17)

Let $J = G_{max}$ and $T = 0$ and go to step 2.

<u>Step 2:</u> Let $TT = [\frac{\bar{f}_J}{J}]$

and set

$T = T + TT$

$$\bar{f}_{J-1} = \bar{f}_{J-1} + [\bar{f}_J - TT \times J] \qquad (3.18)$$

Step 3: Let $J \leftarrow J-1$. If $J = 0$, then, T is the minimum number of planes required. Stop. Otherwise go to step 2.

None of the above rules strictly guaranties the existence of a solution within a given path list. Even if a given path list passed all of these checks, there still may or may not be a solution to BRP contained in that path list. But if one of these rules fails, then, this implies that the given path list does not contain any solution. The major advantage gained by using these rules is that they are simple and are very fast in giving an idea about the size of the path list required to achieve a solution.

3.7.3 The Second Step: Search

At this step our objective is to find a solution that satisfies the constraints of BRP (if such a solution exists). As it will be proven in Section 3.8 if there exists a solution, then, it will be the optimal solution to the BRP.

The procedure developed here intends to find out a feasible solution which utilizes P or less planes (i.e., paths) within given the path list. As a result each plane will be assigned to a path. During this step, a partial solution is generated and paths are included in or deleted from this partial solution iteratively until a complete feasible solution is obtained or, otherwise the procedure switches to the Enlargement Step.

The search step can be briefly stated as follows, Step a: Resequence the current sorted list of paths, such that the number of iterations during the search procedure is minimized. That is, perform so called "Blocking" operation.

Step b: Decide on limits of the search.

Step c: Search to find a feasible path to include in the current partial solution. If such a path is found, go to step b. Otherwise delete last path included in the partial solution and go to step b.

The details of these steps will be discussed in the following sections.

3.7.3.1 Blocking The Path List

This step is in principle similar to the first step of the classical SPP algorithm suggested by Christofides; namely the "blocking" step. That procedure was explained in detail while introducing the algorithm. Since classical SPP does not care how many paths (sets) should be utilized and tries to minimize total objective function value, it directly operates on paths which contain uncovered elements of load set L. So the blocking process is designed to group paths which do not cover the same loads (rows) together. But in our case, the number of loads per plane is crucial. If we choose paths which contain more loads, then, we shall have more chance to obtain a feasible solution which utilizes given or less number of planes, quickly. Since we do not care about the total objective function value.

Most of the time there is an unbalanced distribution of load frequencies within the given path list. Although some of the loads covered by many paths, some of them are covered by relatively few paths. Such a situation results from the geographical distribution of loads and planes initially. If there are loads far away from the initial airports of the available planes, then too much time has to be spent to carry such loads. While, in the mean time loads in the vicinity of the initial airports of the planes can be covered by several combinations. Often the maximum distance to generate paths (d_{max}) is set such that these remote loads are just covered. So the blocking procedure should enable the search mechanism to focus on such remote loads first.

The foregoing discussion reveals that there are two important decision criteria in the blocking process.

i) The frequencies of loads.

ii) The load cardinalities of paths.

Thus, depending upon the nature of time data of the original problem and the information generated while taking the paths from mass storage, two alternate methods for blocking can be identified.

The next two sections will describe the details of these methods. But before getting in them, note that as a result of blocking process the sequence of paths in the current paths list is changed, that is a new path list will be generated, the search for a path to assign a plane will start from the bottom of this new path list and will gradually move up (See Fig. 3.3). Therefore critical paths should be forced to the bottom portion of the new list.

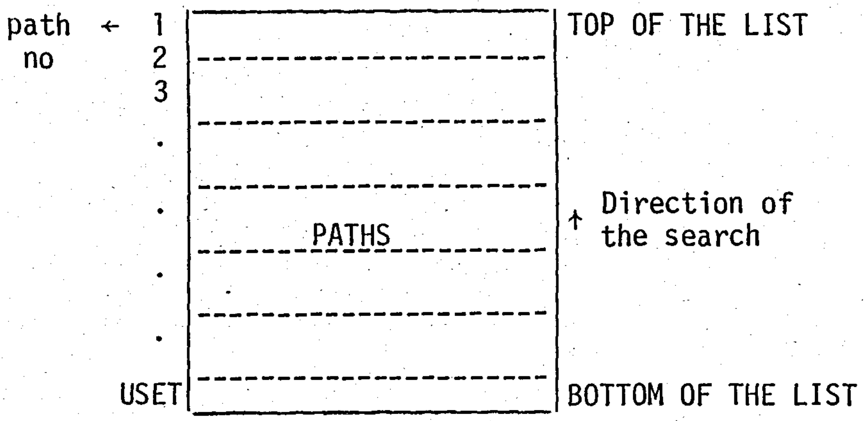


FIGURE 3.3 - The path list

3.7.3.1.1 *Method A: Blocking With Respect to Load Frequencies*

Usually when there are remote loads to be carried, there are greater deviations in the load frequencies. This type of blocking is preferable when there are such deviations. The paths covering loads whose frequencies are the least, are selected and located to the bottommost empty positions of the new list sequentially as follows.

<u>Step a:</u> Calculate total load frequencies, i.e.,

$$T_{\ell} = \sum_{g=1}^{G_{max}} f_{g,\ell} \qquad (3.19)$$

Note, T_ℓ is the total number of times load ℓ is covered in the given path list.

<u>Step b</u>: Choose an unconsidered load whose frequency is the least, Ties are broken arbitrarily.

<u>Step c</u>: Scan the path list to find out paths which cover the chosen load and not marked yet. If any such path found, then, put them at the bottommost available locations of the new path list and mark them on the old path list.

<u>Step d</u>: If all loads are considered, then stop. Otherwise go to step b.

During the insertion of the paths into the new path list (in step c), we have alternative ways to proceed, since there are in general more than one candidate paths selected from the path list which cover the same load. The problem is to choose the one which will be located to the bottommost available position in the new path list, so that the chosen one will be considered first by the search mechanism. The best approach appears to be to locate the path whose load cardinality is the largest to the bottommost available position. In order to avoid the use of additional memory space, a heuristic rule is adopted when coding the algorithm. According to that rule, the longest path among the candidate paths is located at the bottommost position, based on the generation that the longer the path, the larger the number of loads it will cover.

3.7.3.1.2 *Method B: Blocking With Respect to Load Cardinalities*

This type of blocking is preferable when there are no great deviations in the load frequencies. Under such circumstances the method suggested previously would have no significant effect. Thus, in such cases, it would be better to locate paths which cover more loads to the bottom portions of the new path list, so as to avoid unnecessary iterations of the search procedure with paths covering relatively few loads. This type of blocking can be summarized as follows.

<u>Step a:</u> Set $g = G_{max}$

<u>Step b</u>: Choose paths from the path list which covers g loads and locate those paths at the bottommost available places of the new path list.

<u>Step c</u>: Decrease g by one. If $g = 0$, then, stop. Otherwise go to step b.

Also, in this type of blocking we have alternative ways to proceed in second step. In this case, the best is to locate the path of least length at the bottommost available position.

3.7.3.1.3 *Storage Space for The New Path List*

Although a new path list is generated, it is not necessary to store it as a new list. An imaginary path list will be enough. Such that only a one-dimensional array can be utilized instead of a new list. The elements of this array are pointers representing the paths on the new list. Each element points to the position of the related path in the old path list. This array is called the ADRES array.

3.7.3.2 Deciding on The Limits of The Search

This is an iterative step. Each time the procedure reaches this step, we have a partial solution which utilizes $0 \leq KPL < P$ planes (KPL being the number of planes utilized in that partial solution). After this step, the procedure will begin to search for a path to assign the $(KPL+1)^{th}$ plane. Note, the search procedure will move upwards from bottom search limit to top search limit. Let us denote the bottom search limit by KSET and the top search limit by KRT.

Depending on the number planes (KPL) utilized in the current partial solution, the search limits can be established as follows.

A) Assignment of First Plane (KPL = 0)

This means either the process achieves to this step for the first time or one wants to change the path assignment of the first plane.

a) The top search limit (KRT)

The top search limit varies by the blocking method utilized. Let us first consider the case for the blocking method A.

Let SF_J be the path number of the first occurrence of load J in the new path list (i.e., load J has not been occurred in paths indexed $1,2,\ldots,(SF_J-1)$). Then, the top search limit will be,

$$KRT = \max_{J\epsilon\{1,..,M\}} (SF_J) \tag{3.20}$$

Since the load which gives rise to the value of KRT, does not occur in any of the paths $1,2,\ldots,(KRT-1)$, it will be useless to search these paths in order to assign the first plane. That is, if we cannot assign the first plane to path whose index is greater than or equal to the value of KRT, then, this implies that we cannot find a solution to BRP. Let us now consider the case of blocking method B.

Let YF_J be the path number of the first occurrence of path with load cardinality J, in the new path list (i.e., load cardinalities of the paths $1,2,\ldots,(YF_J-1)$ are less than J). Recall that $K = (M-QxP)$, which is the number of planes required to assign to paths with load cardinality $(Q+1)$, given there is none with load cardinality $(Q+2)$. Depending on the value of K, there can be two cases.

i. $K > 0$. Assume that no plane assignments have been realized until $(YF_{M1}+K)^{th}$ path. Then, exactly K planes should be assigned to paths indexes $(YF_{M1}+K-1)$, $(YF_{M1}+K-2)$,.. ..., YF_{M1}. Since all paths with load cardinalities $(Q+2)$ are exhausted, due to the blocking method B. Therefore assigning the first plane to path whose index less than $(YF_{M1}\ K-1)$ will not enable other $(K-1)$ planes to be located on that interval and will cause infeasibility. Thus, the top search limit is determined as,

$$KRT = YF_{M1} + K - 1 \tag{3.21}$$

ii. K = 0. Similar to the reasoning in the previous case, all P planes should be assigned on the interval $(YF_{M1}+P-1),(YF_{M1}+P-2),\ldots,YF_M$, if no plane assignments has been realized up to that point. Thus, the top search limit will be,

$$KRT = YF_{M1} + P - 1 \tag{3.22}$$

b) The bottom search limit (KSET)

Initially the bottom search limit for the first plane is the bottommost path. Namely KSET = USET (USET being the total number of paths in the path list). If the process reaches this step again since KPL = 0, then, the bottom search limit has to be the path which is just above the current bottom search limit, i.e., KSET is substituted by (KSET-1).

Since the partial solution set is empty (KPL = 0) at this stage, no search is performed to assign the first plane and that plane is assigned to (KSET)th path. Furthermore, the value of KRT does not change (as far as the path list is fixed) throughout the iterations when KPL = 0. Let LSET be the value of KRT when KPL = 0 Therefore there are (USET - LSET + 1) possible paths to assign the first plane. If (USET - LSET + 1) passes over this step, result without a feasible solution, then, the given path list cannot contain any feasible solution. We terminate the search over this path list, since block is exhausted. We move to the Enlargement step.

B) Plane Assignments Beyond the First Assignment
$(1 \leq KPL < P)$

a) The top search limit (KRT)

If some planes have been assigned already, then there are two candidates for the top search limit.

i. Let; KLD: number of loads covered by KPL planes.

$RL = M - KLD$: number of uncovered loads.

$RP = P - KPL$: number of unused planes.

Then, YUK = smallest integer $\geq RL/RP$ (3.23)

YUK is a redefinition of M1 for the reduced problem. Then it means at least one plane should be assigned to a path whose load cardinality is at least YUK in order to achieve feasibility. Hence, one candidate for the top search limit will be,

$$MRT = YF_{YUK} \tag{3.24}$$

That is, $(KPT + 1)^{th}$ plane cannot be assigned to paths 1,...,(MRT-1). Assume $(KPL+1)^{th}$ plane has been assigned to a path KP whose index satisfies $1 \leq KP < MRT$. Then, the next plane should be assigned the path whose index less than KP. But load cardinalities of all those paths are less than YUK. Hence, there is no way to reach a feasible solution.

ii. The second candidate for the top search limit: S

$$LRT = \max_{i\varepsilon[\text{all uncovered loads}]} (SF_i) \tag{3.25}$$

The (KPL+1)th plane should cover at least one of the uncovered loads. Specifically this plane should either cover the load whose first occurrence is at the bottommost or be assigned to a path below that level so as to enable the next plane to cover the load under discussion. Note the proofs of the two candidates are complementary.

Since any violation of these two candidates will cause infeasibility, the top search limit will be the maximum of them. That is,

$$KRT = \max[LRT, MRT] \tag{3.26}$$

b) The bottom search limit (KSET)

i. If the last operation is a path addition. Let us first consider the case when blocking method A is used. Let E be the set of all loads covered by the last plane included in the partial solution and define,

$$KZ = \max_{i\varepsilon E} (SF_i) \tag{3.27}$$

Then, set KSET to KZ-1. This is an implicit enumeration of all the paths between old and new KSET values. Since these paths should necessarily contain the load which gave rise to the value of KZ, the search on these paths will always be useless.

Let us now consider the case when blocking method B is used. Let the last plane assigned to path KP, where KSET $\leq$ KP $\leq$ KRT. Then, the new KSET values should be (KP-1). The search for the next plane should start just after the path KP.

ii. If the last operation is a path deletion.

Let KR be the path number which is deleted. Then KSET will be (KR-1). That is, bottom search limit is set to the path which is just above the deleted path.

After the search limits are determined, we check whether KRT < KSET. If so, then, this means the feasible region to assign $(KPL+1)^{th}$ plane is collapsed and the current partial solution cannot lead to a feasible solution anymore. Therefore, we must drop KPL^{th} plane and go back to determine the top and bottom limits of the new search for a feasible path. Otherwise, we can go on searching a path to assign $(KPL+1)^{th}$ plane between these limits.

In Fig. 3.4, the computation of search limits are illustrated.

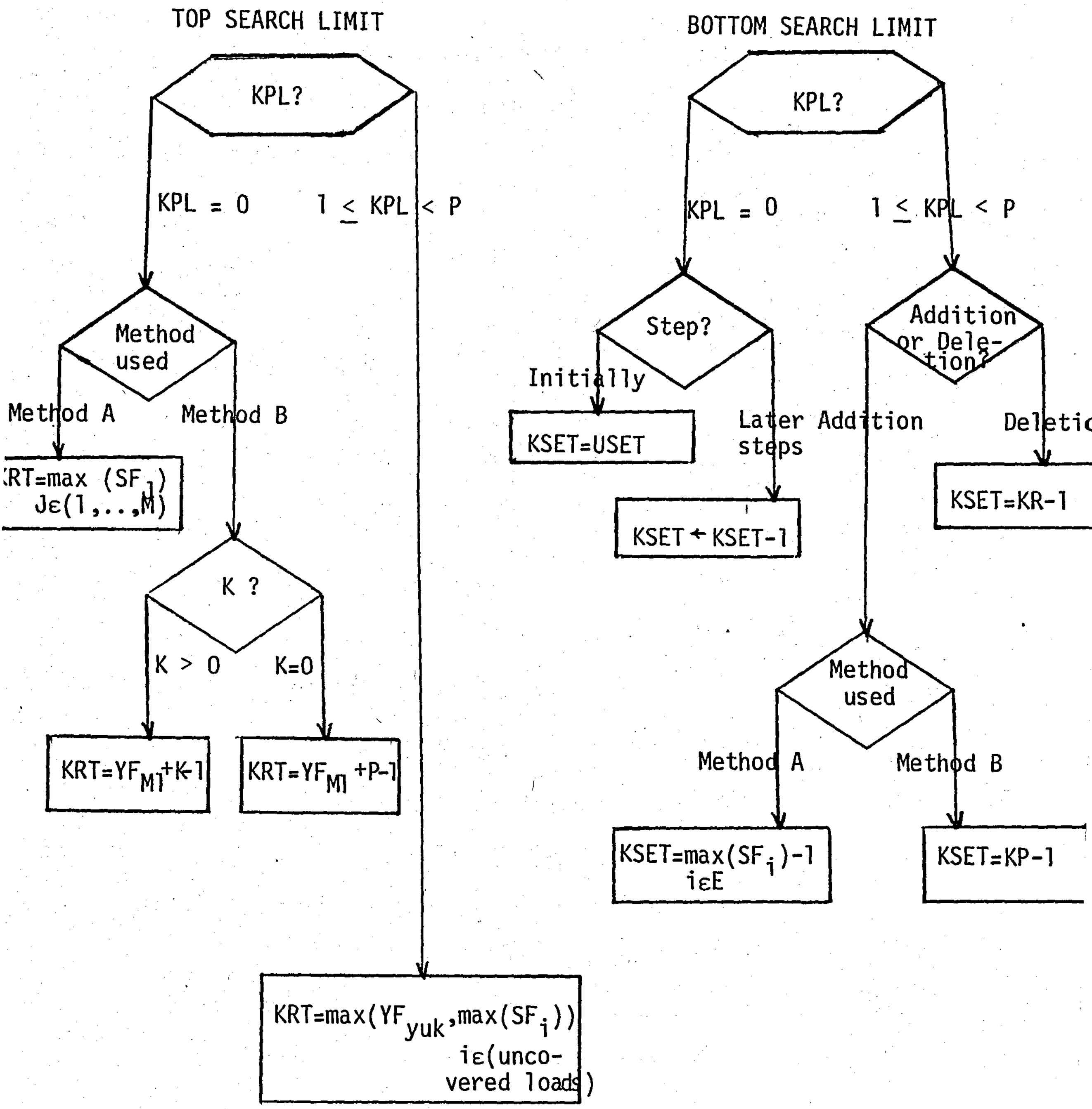


FIGURE 3.4 - The flowchart of the computation of search limit

3.7.3.3 Search For a Feasible Path

The objective of this step is to assign (KPL+1)th plane to a path which is between predetermined limits. The search for that path will start from the bottom search limit and move up by checking each path individually for feasibility. First thing to check is whether the candidate path originates from an airport whose planes have been used already by the current partial solution. If such a case occurs, then, that path should be omitted and the one just above it, has to be checked. After the initial airport constraint, we should check whether the candidate path covers any of the loads that are already covered by the current partial solution. Also, such paths should be omitted due to so-called "no-overcovering" restriction of BRP. So, the process gradually moves up to the top search limit.

If one of the paths satisfies the constraints given above, then, (KPL+1)th plane can be assigned to that path, and that path can be added to the current partial solution. Since we have made the (KPL+1)th assignment, we can set KPL as (KPL+1) and enlarge the partial solution at hand.

At this stage, if the total number of loads covered by the current partial solution equals to the total number of loads to be covered, then, we terminate having found a feasible solution satisfying all the constraints. This solution is the optimal solution to the BRP. The proof of optimality will be given in Section 3.8. However, KPL need not be equal to the total number of planes available. We have the possibility of reaching a solution using fewer planes than available (See Example D).

On the other hand. If the total number of loads covered is less than the total number of loads to be covered, then we must redefine the search limits for the next plane and continue search process as defined.

While searching for a path, if non of the paths satisfy the feasibility constraints that is, paths between KSET and KRT are exhausted, then, we have to delete the KPLth plane from the partial solution and again continue by redefining search limits.

3.7.4 The Third Step: Enlargement

If no feasible solution has been found at the end of the search step (i.e., block one is exhausted), then, we shall take some more paths from mass storage into the current path list so as to enlarge the path list. Let ℓ_0 be the length of the longest path in the path list in which no feasible solution has been detected.

At this point we know that the length of paths remaining in the mass storage are greater than ℓ_0, since we must have taken all such path in previous steps. Let ℓ_1 be the length of the shortest path in the mass storage. By the previous discussion we know $\ell_1 > \ell_0$. Then, we should scan mass storage until we catch a path whose length is greater than ℓ_1, and we shall enlarge the path list by taking in all paths of length ℓ_1. After the enlargement, we switch back to the beginning of Search step in order to reinitiate blocking process.

One must note that the original time data is assumed to be integer. So that after every enlargement operation, the number of paths taken into the path list will not be equal to one most of the time.

3.8 THE OPTIMALITY OF SOLUTION FOUND AT THE SEARCH STEP

While discussing the search step, we have claimed that if one achieves a feasible solution, then that solution will be the optimal solution to the BRP. The following proposition will give the proof of that declaration.

Propositon 3.6. The feasible solution found by applying the Bottleneck Routing Algorithm is the optimal solution to the BRP.

Proof: The search step can be achieved from either the Initialization Step or the Enlargement Step.

In both of the cases, we have a path list, and let ℓ_0 be the length of thelongest path in this path list. In any case guarantee that there is no feasible solution to BRP in some path list. Then, we enlarge that path list by taking all paths of length $\ell_0+\varepsilon$ from the mass storage. During this process we make sure that there is no path of length ℓ, such that $\ell_0 < \ell < \ell_0+\bar{\varepsilon}$. Thus, if we are able to find a solution in this enlarged path list, then, it should be the optimal solution of BRP with the optimal value being $(\ell_0+\varepsilon)$. The optimal value of BRP cannot be reduced further. Since, if it can be reduced, then, it should be ℓ_0. But there is no solution among paths whose length is less than or equal to ℓ_0.

Q.E.D.

3.9 GENERATION OF ALTERNATIVE OPTIMAL SOLUTIONS TO BOTTLENECK ROUTING PROBLEM

If the algorithm is not terminated after finding the optimal solution of BRP, then it can generate all other alternative optimal solutions of the problem. This can be achieved by assuming of feasible (optimal) solution at hand as a partial solution. In this case the algorithm will terminate at the point where all the paths that first plane can be assigned are exhausted (i.e. the first block is exhausted). In this way, the procedure will turn to be a full enumeration process. Since bottleneck objection function value does not change during the process. The solutions generated by this way will be alternative optimal solutions to the BRP.

If there are more than one load specified between any two airports, then imaginary alternatives will be produced by the procedure.

To illustrate this fact better, note that load 4 and 10 in Example B refer to loads to be carried from airports 4 to airport 3. The optimal solution found at the end of algorithm assigns plane 1 to loads 7,10, and 6 and plane 3 to loads 4, 8, 9 and 1. Thus, if we do not terminate the process, we eventually reach a solution identical to the previous one, except that the positions of load 4 and 10. That is, plane 1 will be assigned to loads 7, 4 and 6 and plane 3 will be assigned to loads 10, 8, 9, and 1. When there are many loads in this status (See Example F) the number of imaginary alternatives will blow up.

3.10 THE ALTERNATIVE OPTIMUM TO THE BOTTLENECK ROUTING PROBLEM WHICH RESULTS IN THE MINIMUM TOTAL COST

There are case where, besides bottleneck objective function value, the value of the total job time is also important. That is, the first objective is to minimize the maximum job time and the second objective is to minimize the total job time.

The SPP approach applied by Ulusoy, by definition catches the solution defined above, since the SPP algorithm tries to minimize the total cost.

The algorithm developed here can be adopted in the following manner for obtaining that solution.

Let Z be the total objective function value that corresponds to the partial solution and $\bar{Z}$ be best total objective function value yet reached among alternative optimal solutions of BRP. Initially, $\bar{Z}$ is set to infinity since there is no feasible solution yet.

While testing path KP to assign (KPL+1)th plane, first check if Z+(length of path KP) $< \bar{Z}$. If so, continue testing. Otherwise, omit the path KP.

Continue the process as defined in Section 3.9 until the first block is exhausted. Thus, the solution obtained in this way will be alternative optimum to BRP which results in the minimum total cost.

This process cannot implicitly enumerate some solutions for undesirable Z values due to the nature of blocking, but computational experience has shown that the algorithm is not so poor compared to SPP in that respect.

3.11 DEMONSTRATION OF THE ALGORITHM

Example C is chosen to demonstrate the steps of the algorithm. Most of the stopping rules are inactive in this example. But the solution is found easily among 72 paths only.

The maximum distance criterion applied to the generated paths is 150 and this resulted in a total of 163 paths after the elimination process (See Appendix D).

From the data we get,

$$M = 12, \ P = 6, \ RP = 2, \quad Q = M / P = 2$$

$$K = 0, \text{ therefore } M1 = 2$$

The nodes of the transformed network represent the following (referring to the formulation in Section 3.1):

node 2 → refers to airport 3.

node 3 → refers to airport 4

node 4 → refers to load 1 (from airport 1 to airport 2)

node 5 → refers to load 2 (from airport 4 to airport 1)

node 6 → refers to load 3 (from airport 2 to airport 4)

node 7 → refers to load 4 (from airport 4 to airport 3)

node 8 → refers to load 5 (from airport 4 to airport 2)

node 9 → refers to load 6 (from airport 3 to airport 1)

node 10 → refers to load 7 (from airport 3 to airport 4)

node 11 → refers to load 8 (from airport 3 to airport 2)

node 12 → refers to load 9 (from airport 2 to airport 1)

node 13 → refers to load 10 (from airport 4 to airport 3)

node 14 → refers to load 11 (from airport 1 to airport 3)

node 15 → refers to load 12 (from airport 1 to airport 4)

BRP algorithm starts by reading the paths from mass storage sequentially and after each length change it goes on the stoppping rules.

Rather than explaining each step of the algorithm we shall just briefly summarize the situation (detailed output is available in Appendix D).

First note that, node 14 is not covered until 63rd path. So there cannot be feasible solution up to this point. But we cannot go on applying stopping rules just after reading 63rd path. We have to wait until a length change occurs. Note that the length of path 72 is 120 and length of path 73 is 125. Therefore we can apply stopping rules at this point with the frequency matrix given in Table 3.3.

Since $f_{11} = 12 =$ total number of loads. Rule 1 is satisfied.

Since $G_{max} = 3$. Rule 2 is also satisfied.

Rules 3 and 4 are inactive since $G_{max} > M1$.

For minimum plane check, note,

5 of loads are covered by paths of load cardinality 3.

6 of loads are covered by paths of load cardinality 2.

1 of loads are covered by paths of load cardinality 1.

At our best, we can assign a plane to a path of load cardinality 3 and 2 loads will remain to paths of load cardinality 2. Thus, 8 loads can be covered by paths of load cardinality 2. So, at our best we can handle these loads by 4 planes and the last load should be carried alone. Therefore, we need a total of 6 planes. Since we have already 6 planes, the rule is satisfied and the procedure switches to the search routine.

The blocking step:

In accordance with the earlier analysis "Method A" for blocking will be adopted here since there exists an unbalanced distribution of load frequencies. So, we first should choose the load which is least covered. That is the load 11 (node 14), it is covered only by 63rd path, therefore we should put this path at the bottommost position of the new list. Namely, ADRES(72) = 63.

The next candidate load is 12 (node 15) which is covered only 3 times. The paths covering load 12 (node 15) are:

69: 2 - 15

44: 3 - 5 - 15

30: 3 - 15

Therefore, starting from the bottom of both lists we shall locate these paths. Namely,

ADRES (71) = 69

ADRES (70) = 44

ADRES (69) = 30

Indeed the generalization accepted in Section 3.7.3.1.2 did not hold in this case. It is preferable to locate 44th below the 69th path since it covers more loads. But counting the number of loads in each path and organizing ordering accordingly can be easily done.

Then, depending upon the total frequencies the rest of the add array between two lists can be generated as listed in Appendix D.

The first occurrence of loads and load cardinalities will be obtained by the process defined in the Flowchart-3 and they are shown in Tables 3.4 and 3.5 respectively.

The Search Step:

Since we have 72 paths, KSET is also 72 initially. Since node 14 is only covered in 72, first plane has to be assigned on the 72nd path (namely the 63rd path in the original list). There cannot be any solution which does not cover that path.

If path 71 had also covered node 14 then, KRT should be 71 and the first plane should be assigned, either 72 or 71. But that is not the case.

As a result first plane is assigned to path 72 and at this point search limits can be established as:

$$\begin{aligned} KSET &= \max_{J \varepsilon [\text{loads covered in } 72^{nd} \text{ path}]} SF_J - 1 \\ &= \{\max\ [72]\} - 1 \\ &= 71 \end{aligned}$$

KRT = 69 That is, the point at which node 15 (load 12) has first occurred. That is, path 69 contains load 12. Paths 70, 71, 72 may contain load 12 and paths 1,...,68 do not contain load 12.

Formally KRT calculated as follows,

RL = 12 - 1 = 11

RP = 6 - 1 = 5

which implies YUK = 3.

Thus, $MRT = YF_3 = 12$

and $LRT = \max_{i\varepsilon[1,2,3,4,5,6,7,8,9,10,12]} [SF_i] = 69$

Therefore, $KRT = \max[16,12] = 69$.

According to this result only three paths have to be checked for assigning the second plane. Namely, paths 71, 70, and 69. Path 71 involves nodes 2 and 15 which is feasible for the current partial solution. The algorithm decided to assign the second plane to the 71st path. It immediately follows that KSET = 68 $KSET = SF_{15} - 1$. Then, we have two candidates for KRT as explained previously,

$$RL = 12 - 2 = 10$$

$$RP = 6 - 2 = 4$$

which implies YUK = 3.

Thus, $MRT = YF_3 = 12$

and, $LRT = \max_{i\varepsilon[1,2,3,4,5,6,7,8,9,10]} [SF_i] = 65$

Therefore, $KRT = \max[65,12] = 65$.

Hence, the search limits for the third plane are established to be 68 and 65. Within this interval path 68 is chosen which results KSET = 64 and KRT = 58. From this interval path 64 is chosen which results KSET = 57 and KRT = 52. At this point path 57 is chosen and the current partial solution become 72, 71, 68, 64, 57. But this resulted KSET = 51 and KRT = 72. Since search limits overlap we must drop path 57 and move up to search for another path. Then, path 55 will be the path which satisfies the constraints.

Similarly the process continues until the partial solution contains 72, 70, 68, 64, 35, 22. At this point all loads are covered to achieve a solution which is feasible and so optimal to the bottleneck routing problem. The details of the process are shown in the computer output at Appendix D.

TABLE 3.3 - The Frequency Matrix After Getting 72nd Path Into the Path List

		N o d e s														
Meanings of nodes			Port 3	Port 4	Load 1,2	Load 4,1	Load 2,4	Load 4,3	Load 4,2	Load 3,1	Load 3,4	Load 3,2	Load 2,1	Load 4,3	Load 1,3	Load 1,4
Node Number		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Load cardinality	1	12	1	12	2	2	2	2	2	2	2	2	2	2	1	2
	2	11	20	24	5	7	9	12	9	2	13	11	8	12	0	1
	3	5	2	3	0	0	0	3	2	0	5	2	0	4	0	0
Totals		28	23	39	7	9	11	17	12	4	19	15	10	17	1	3

TABLE 3.4 - First Occurrence of Loads in The New Path List

Load	First occurrence Path No.
1	58
2	52
3	32
4	7
5	24
6	65
7	1
8	14
9	43
10	3
11	72
12	69

TABLE 3.5 - First Occurrences of Load Cardinalities in The New Path List

Load Cardinality	First Occurrence Path No.
1	1
2	5
3	12

IV. THE AIRPORT CAPACITY CONSTRAINTS

4.1 INTRODUCTION

Once the routes of the cargo planes have been determined, the longest mission time is defined by the objective function value of the related solution of the Bottleneck Routing Problem (BRP). At this point, the service facilities at the airports should be checked to see whether they are sufficient not to cause any delay to any of the flights. The optimal solution value of the BRP constitutes of deadline for all planes to finish their job. It is obvious that at least one of the planes will complete its job just at the deadline. Other planes will have some slack time to complete their jobs. The slack times can be utilized, if there is demand for service at any airport beyond its capacity.

As it was mentioned in Chapter I there are two types of constraints on service facilities at the airports.

i. The number of planes that are serviced at a given airport at any time cannot exceed some predefined limit. Let CAP_i be the maximum number of planes that can be serviced (loading or unloading) at any instant at airport i.

ii. The planes which have enough slack times can join the queues at some of the airports, if service facilities are busy. But there is also a limit on the size of the queue at each airport. Let $QCAP_i$ be the maximum queue size at airport i at any instant.

Throughout this chapter we shall try tofind out a schedule for planes which satisfies these constraints.

In this study the service times of the planes at the airports are assumed to be constant. That is, these times are independent of both planes and airports. The service time of a plane at an airport has two components.

i. TL: the loading time

ii. TU: the unloading time.

Although these components of the service time are assumed to be constant, this is not a strict requirement. They may vary from airport to airport and from load to load. If such a situation exists, then different times can be added on arcs defined in the network formulations. For the sake of simplicity in notation and for the sake of memory size requirement in the computer applications, service times are assumed to be independent of loads and airports.

Before getting in the details of the technique developed to handle such constraints, one must note the following definitions.

i. MIS_i is the mission completion time of the i^th plane, if all planes start at the same time and if no delay occurs during that mission times. Their values are obtained from the given solution of BRP. Specifically, they are the lengths of paths that the related planes are assigned to.

ii. DUE is the longest mission time. That is,

$$DUE = \max_{i \in \{1,\dots,P\}} (MIS_i) \tag{4.1}$$

This DUE is the objective function value of the BRP solution and corresponds to the deadline.

iii. $SLACK_{i,t}$ is the amount of remaining slack time for plane i at any time t and is defined by the equation,

$$SLACK_{i,t} = DUE - MIS_i - WAIT_{i,t} \tag{4.2}$$

where $WAIT_{i,t}$ is the total time spent at queues by plane i up to time t (i.e., the idle time).

4.2 n-JOB, m-MACHINE JOB SHOP SCHEDULING PROBLEM

The scheduling of planes with given fixed routes has great similarities with the n-job, m-machine job shop scheduling problem. So it will be better to define this problem first.

Suppose we have n jobs $J_1,\dots,J_n$ and m machines $M_1,\dots,M_m$ which can handle at most one job at a time. Job J_i $(i = 1,\dots,n)$ consists of a sequence of n_i operations O_r, each of which corresponds to the processing of job J_i on machine $\mu(O_r)$ during an uninterrupted processing time of P_r time units. We seek to find a processing order on each machine such that to optimize the choosen measure of effectiveness (Conway, Miller and Maxwell, 1967).

This problem is quite formidable. Major difficulties are computational since there are $(n!)^m$ possible schedulings in general form. There are no efficient exact solution procedures known. Conway, Maxwell and Miller (1967) have formulated integer programming models but computational results are not encouraging.

Some heuristic models are used in general job shop scheduling problem. Most commonly used procedures known as "dispatching rules". These are simply logical decision criteria that enable an analyst to select next job for processing at a machine when that machine becomes available. Thus, scheduling decisions are made sequentially over time instead of all at once. Such procedures always include the concept of "job priority". A job priority is a numerical attribute of a job, defined in such a way that, a job with the smallest priority is scheduled first. Most of the time these priorities are assigned heuristically, and most of time various types of information available, about the status of work centers, are incorporated in these decisions (Johnson and Montgomery, 1974).

4.3 THE RELATIONSHIP BETWEEN PLANE SCHEDULING AND n-JOB, m-MACHINE JOB SHOP SCHEDULING PROBLEM

The scheduling of planes with fixed routes is a complicated version of the n-job, m-machine scheduling problem, where the jobs are the planes and the machines are the airports. Since the sequence of loads that should be carried by each plane is known, we can easily extract the sequence of airports which should be visited. Thus the operation sequences are defined.

The operation times are defined as loading and unloading times depending whether the plane is waiting to load or unload respectively.

Each plane enters the system exactly once and should complete its time at time referred to as DUE in Eq. (4.1). At the end of scheduling process all the planes should complete their job latest at this time. Otherwise the objective function value of BRP will be increased by the maximum delay amount.

These similarities imply that scheduling of planes is not so easy. Furthermore, we have following differences from n-job, m-machine scheduling problem causing additional difficulties.

i. We talk about the existence of a sequence dependent set-up time, since the planes spend time while flying between airports, if we represent flight times as set-up times.

ii. Each machine can handle more than one job at any instant. That is each airport k is capable of servicing at most CAP_k planes at any instant (See Fig. 4.1).

iii. Queues are limited. That is, each airport k has a queue capacity of $QLAP_k$.

iv. The sequence of the operations on a job can be altered since the routes of planes can be changed if they still complete their mission within the optimal value of problem.

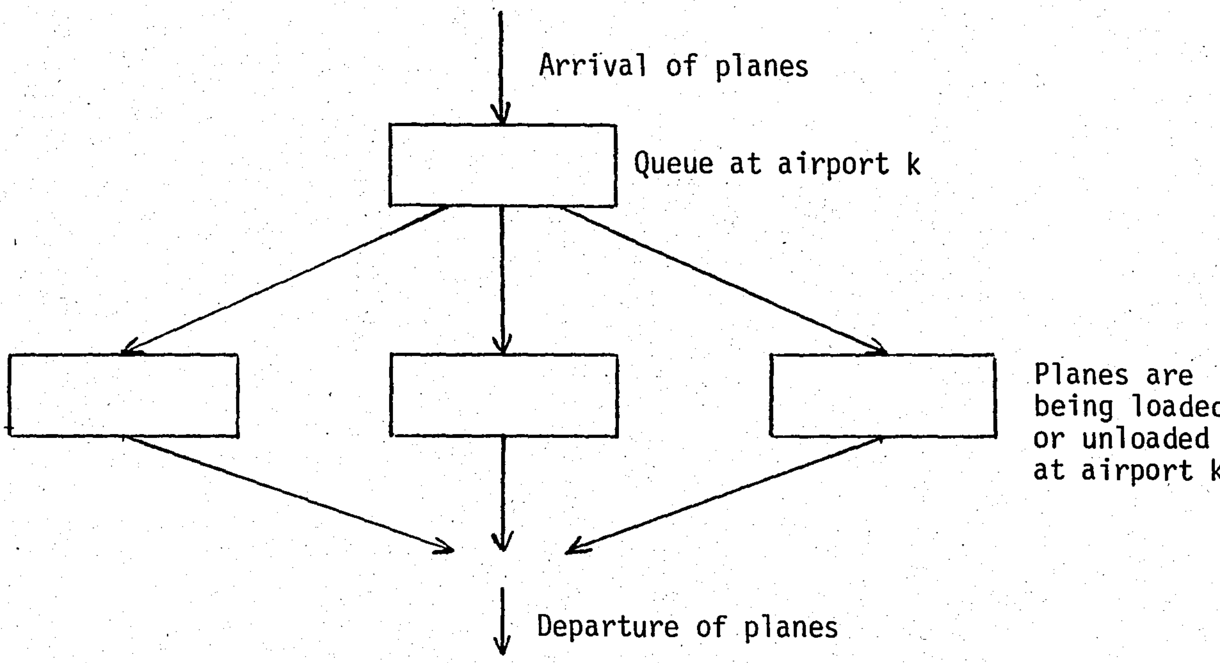


FIGURE 4.1 - Service mechanism at airports

Formulating the plane scheduling problem as a job shop scheduling problem, establishes that it is very hard to develope an exact solution procedure in here. Hence we must resort to heuristic procedures, namely dispatching rules.

Once the use of job shop scheduling heuristic is accepted, the problem is to decide what the priorities of planes should be, if a queue occurs.

In plane scheduling case these priorities should take into account the remaining slack times of the planes. The plane which has the least slack time should be scheduled first, since the penalty cost of not satisfying the given DUE time is very high, namely that of the rejection of the solution.

4.4 THE HEURISTIC PROCEDURE DEVELOPED FOR PLANE SCHEDULING

The heuristic procedure developed for plane scheduling is a generalization of the heuristic procedure used for job shop scheduling problems. The major steps of the algorithm can be stated as follows:

Step 1: Initialize Clock

For each plane i (i = 1,...,P), if the initial airport of plane i coincides with the starting airport of first load carried by that plane, then put plane i to the queue of that airport and set its job completion time to zero. Otherwise, set plane i flying from initial airport to starting airport of first load and set the job completion time to the end of this flight. Go to step 5.

Step 2: Update Clock

If all planes completed their mission (i.e., all loads are carried to appropriate airports), stop. Otherwise, set the clock to the minimum of all the job completion times.

Step 3: Update Slack Time

Reduce the slack times of planes which are waiting at the queue by the amount of time spent between previous and current clock times. If for any plane the slack time is negative, then no feasible solution exists, stop. Otherwise, continue.

Step 4: Update Plane Status

For each of the plane i, whose job completion time equals to clock;

i. If plane i finished a flight, then put this plane to the queue of the next airport it has arrived at and set its job completion time to infinity.

ii. If plane i finished loading, then set plane i carrying that load to appropriate airport and set its job completion time to end this flight.

iii. If plane i finished unloading and a loading will follow, then put plane i in the queue of the related airport and set its job completion time to infinity. If plane i finished unloading and an empty flight will follow, then set this plane flying to the appropriate airport and set its job completion time to the end of that flight.

Step 5: Decide on Priorities

For each of the airport J (J = 1,...,R). If unused service capacity of airport k is greater than or equal to the queue, then start processing all planes waiting at that airport. Otherwise, select the planes according to the minimum remaining slack time priority rule. Update all job completion times either to an end of loading or to an end of unloading, depending upon the status of the related plane. If the queues at any airport exceeds the queue capacity, then no feasible solution exists, stop. Otherwise, go to step 2. (Note, precessing a plane can start only if the related job completion time equals to current clock time.)

In practice one would expect that a plane unloaded is loaded without re-entering the queue. But this seperation of jobs is often useful while scheduling. If the seperation of jobs implies some additional time (i.e., that of pulling the plane to the queue or vice versa), then those times can be added to the job completion times of the planes.

4.5 THE HEURISTIC SCHEDULING PROCEDURE AND THE BOTTLENECK ROUTING ALGORITHM

While discussing the solutions of the Bottleneck Routing Algorithm, it has been mentioned that there are some alternative solutions generated. These solutions are sequentially checked by the heuristic scheduling procedure in order to determine whether any of them is feasible, i.e., satisfy the capacity constraints of the airports. The heuristic scheduling procedure does not guarantee a result and it can terminate without a feasible solution. But, if a feasible solution is obtained, then we can accept it as a global optimal solution to the overall problem, since all the constraints are satisfied and the objective function value cannot be reduced further. On the other hand, if scheduling terminates without a feasible solution, other solutions for the bottleneck routing problem must be generated and checked.

While checking the solutions of the Bottleneck Routing Algorithm, following points must be kept in mind:

i. Some of the solutions of the Bottleneck Routing Algorithm are copies of each other, because of the multiple loads between the same airports, as discussed in Section 3.9.

ii. Some of the alternative solutions of the BRP are not generated by the Bottleneck Routing Algorithm, because of the path elimination process. Assume that the path P_1, $[p, \ell_1, \ell_2]$ is of length t_1, and the path P_2, $[p, \ell_2, \ell_1]$ is of length t_2, and $t_1 < t_2 <$ DUE. Since both paths cover same loads and the initial airports are the same, the one which is longer (namely the second one) was eliminated by the path elimination procedure. So the path P_2 cannot appear in any of the alternative optimal solutions, although it can appear in any of the solution where path P_1 appears. That is, if BRA decides to assign a plane to path P_1, then this implies that there exists an alternative solution in which that plane can be assigned to path P_2. Thus, the sequence of airports that should be visited by a plane can be altered. In case of job shop problem one can view this phenomenon as changing the sequence of operations on a job.

iii. The possibility of reaching feasibility increases when the slack times of planes increase. The solution alternatives whose total cost is less will have more slack time in plane routes since length of longest tour does not change. Thus, such solutions have higher possibility of having a feasible solution in sense of airport capacity constraints.

By the help of these ideas the updated version of the Bottleneck Routing Algorithm has been designed where a heuristic scheduling procedure has been added to it as a subroutine. Each time a solution is generated by the Bottleneck Routing Algorithm, that solution is checked by the heuristic scheduling procedure for feasibilty or airport capacity constraints. This solution cannot be a copy of the previous one, since it has a lower objective total function value and a higher possibility of containing a feasible solution.

Even though all the solutions are checked by this process, still there is the possibility of not finding a feasible solution to the scheduling problem. Then, in this case, the Bottleneck Routing Algorithm is forced to retrieve some more paths from mass storage and search for solutions on this enlarged list of paths leading to an increase in the objective function value.

4.6 AN EXAMPLE TO THE SCHEDULING OF AIRPLANES

Consider Example G with the capacity data given in Table 4.1.

TABLE 4.1 - The Capacity Data of Example G

AIRPORT	NUMBER OF PLANES	SERVICE CAPACITY	QUEUE CAPACITY
1	4	2	1
2	0	3	2
3	0	2	1
4	4	4	1
5	0	1	1

The first optimal solution alternative generated by the Bottleneck Routing Algorithm given in Table 4.2.

TABLE 4.2 - The Optimal Solution to Example G

PLANE	INITIAL AIRPORT	LOADS CARRIED	MISSION TIME
1	4	14 - 17 - 1	305
2	4	15 - 16 - 19	300
3	4	11 - 6 - 12	300
4	4	2 - 8 - 7	290
5	1	3 - 20	305
6	1	4 - 9	285
7	1	5 - 10	280
8	1	18 - 13	235
		TOTAL TIME	2300

If there are no constraints and all planes will start at the same time, then the schedule of the planes will be as shown in Table 4.3. Note that in Table 4.3 at time 129 planes 3 and 8 are in service at airport 3, and at time 130 plane 5 arrives for unloading. Since the service capacity of airport 3 is only 2 planes, plane 5 has to wait for one of the jobs to be completed. But the mission time of plane 5 is already equal to the length of the longest mission time which is 135. So at this point scheduling terminates with no feasible solution. Note that the possibility of holding service of plane 8 is omitted due to the structure of the heuristic. Thus, the second alternative optimal solution generated by the Bottleneck Routing Algorithm is considered which is given in Table 4.4.

Again for this solution the schedule without any constraint on airport capacity is shown in Table 4.5. Note that at time 130 capacity constraint of airport 3 is violated, since there are three planes to service, namely planes 3, 5 and 8. But this problem is solved as shown in Table 4.6 by holding operation on plane 8. In this schedule all constraints are satisfied and the bottleneck objective function value is minimized. So the solution is globally optimal for capacity constrained bottleneck routing problem. Note that this solution is obtained in only 4.3 seconds on UNIVAC 1106 Computer.

TABLE 4.3 - Optimal BRP Solution to Example G

		1 4	2 4	3 4	4 4	5 1	6 1	7 1	8 1
0						EMPTY (1-2)	EMPTY (1-3)		
20		LOAD AT4	LOAD AT4	LOAD AT4	LOAD AT4 30			LOAD AT1	LOAD AT1
40	45	45	45	45		LOAD AT2	45	45	
60		FULL (4-2)	FULL (4-2) 75	FULL (4-3)	FULL (4-2) 75		LOAD AT3	FULL (1-4)	FULL (1-3)
80		UNLOAD AT2	UNLOAD AT2	UNLOAD AT3	UNLOAD AT2	85	95	85	UNLOAD AT3
100	105	105		105		FULL (2-3)	FULL (3-1)	UNLOAD AT4	110
120		LOAD AT2	LOAD AT2	LOAD AT3	LOAD AT2 130	125	UNLOAD AT1	EMPTY (4-5)	
140	150	150	145	150	155	UNLOAD AT3 150		155	LOAD AT3
160		FULL (2-1)	FULL (2-4) 175	FULL (3-4) 175	FULL (2-5)	EMPTY (3-4)	LOAD AT1	LOAD AT5	
180		185 UNLOAD AT1	UNLOAD AT4	UNLOAD AT4	UNLOAD AT5 185	195	185		FULL (3-2)
200	205	210				LOAD AT4		210	
220		LOAD AT1	LOAD AT4	LOAD AT4	LOAD AT5 230		FULL (1-5)	FULL (5-3) 235	UNLOAD AT2
240	250	255	245	245 FULL (4-3)	FULL (5-4)	FULL (4-1)	255		
260		FULL (1-2) 275	FULL (4-5) 275	265	UNLOAD AT4		UNLOAD AT5	UNLOAD AT3	
280		UNLOAD AT2	UNLOAD AT5	UNLOAD AT3 290		285 UNLOAD AT1			
300									
305									
320									

TABLE 4.4 - The Alternative Optimal Solution to Example G

PLANE	INITIAL AIRPORT	LOADS CARRIED	MISSION TIME
1	4	14 - 17 - 1	305
2	4	15 - 16 - 19	300
3	4	11 - 6 - 12	300
4	4	2 - 8 - 7	290
5	1	3 - 20	305
6	1	1 - 10	275
7	1	5 - 13	275
8	1	18 - 4	220
		TOTAL TIME	2270

TABLE 4.5 - Optimal BRP Solution Alternative to Example G

		1	2	3	4	5	6	7	8
		4	4	4	4	1	1	1	1
0						EMPTY(1-2)			
20		LOAD AT4	LOAD AT4	LOAD AT4	LOAD AT4 30		LOAD AT1	LOAD AT1	LOAD AT1
40	45	45	45	FULL (4-3)		LOAD AT2 45	45	45	
60		FULL (4-2)	FULL (4-2) 75		FULL (4-2) 75		FULL (1-5)	FULL (1-4)	FULL (1-3)
80		UNLOAD AT2	UNLOAD AT2	UNLOAD AT3	UNLOAD AT2		95	85	UNLOAD AT3
100	105	105		105	110	FULL (2-3)		UNLOAD AT4 110	
120		LOAD AT2	LOAD AT2	LOAD AT3	LOAD AT2 130	135	UNLOAD AT5	FULL (4-3)	LOAD AT3
140	150	150	145	150	155	UNLOAD AT3	LOAD AT5 150	155	
160		FULL (2-1)	FULL (2-4) 175	FULL (3-4) 175	FULL (2-5)	EMPTY (3-4)		LOAD AT3	FULL (3-1)
180		UNLOAD AT1 185	UNLOAD AT4	UNLOAD AT4	UNLOAD AT5 185		195	195	
200	205	210				LOAD AT4	FULL (5-3)		UNLOAD AT1
220		LOAD AT1	LOAD AT4	LOAD AT4	LOAD AT5 230			FULL (3-2)	
240	250	255	245		FULL (5-4)	FULL (4-1) 250	250		
260		FULL (1-2) 275	FULL (4-5) 275	FULL(4-3) 265	UNLOAD AT4	275	UNLOAD AT3 275	UNLOAD AT2	
280		UNLOAD AT2	UNLOAD AT5	UNLOAD AT3 290		UNLOAD AT1			
300									
305									
320									

TABLE 4.6 - The Optimal BRP Solution Alternative Which Satisfies Capacity Constraints in Example G

		1	2	3	4	5	6	7	8
		4	4	4	4	1	1	1	1
0						EMPTY (1-2)			
20		LOAD AT4	LOAD AT4	LOAD AT4	LOAD AT4 30		LOAD AT1	LOAD AT1	WAIT AT1
40	45	45	45	45		LOAD AT2 45	45	45	
60		FULL (4-2)	FULL (4-2) 75	FULL (4-3)	FULL (4-2) 75		FULL (1-5)	FULL (1-4)	LOAD AT1
80		UNLOAD AT2	UNLOAD AT2	UNLOAD AT3	UNLOAD AT2		95	90	
100	105	105		105		FULL (2-3) 110		UNLOAD AT4	FULL (1-3)
120		LOAD AT2	LOAD AT2	LOAD AT3	LOAD AT2 130	UNLOAD AT3 135	UNLOAD AT5	130 EMPTY (4-3)	WAIT AT3
140	150	150	145	150	155		150 155	WAIT AT3 145	UNLOAD AT3
160		FULL (2-1)	FULL (2-4) 175	FULL (3-4) 175	FULL (2-5) WAIT AT5	EMPTY (3-4)	LOAD AT5	LOAD AT3 170	
180		185 UNLOAD AT1	UNLOAD AT4	UNLOAD AT4	185 UNLOAD AT5				LOAD AT3
200	205	210	205			LOAD AT4	FULL (5-3)	215	
220		LOAD AT1	LOAD AT4	LOAD AT4	LOAD AT5 230			FULL (3-2)	FULL (3-1)
240	250	255	245	FULL (4-3) 250	FULL (5-4)	FULL (4-1) 250	255	255	
260		FULL (1-2) 275	FULL (4-5) 275	270	UNLOAD AT5	275	UNLOAD AT3	UNLOAD AT2	UNLOAD AT1
280		UNLOAD AT2	UNLOAD AT5	UNLOAD AT3 295		UNLOAD AT1			
300									
305									
320									

4.7 DISCUSSION OF THE HEURISTIC SCHEDULING PROCEDURE

The heuristic scheduling procedure can be developed further to handle more alternative ways to scheduling the planes. First of all, other priority rules can be established. Secondly, one can change the previous priorities, if the queue capacity of an airport is violated. Also one can change overall operation sequences. But it is very hard to impose such changes on these types of heuristics because of the extensive computation time and memory size requirements. Indeed there is no serious attempt to change previous priorities or to change operation sequence in literature. But one could possibly introduce some more powerful decision rules. The decisions can be taken by checking events in future more carefully. That is, the future queues can be estimated and more reliable decisions can be taken. Also, priority can be defined as a function of not only remaining slack time but also of the remaining slack time and some other status variables such as remaining number of operations, remaining processing time, etc.

V. A SUGGESTION FOR FURTHER RESEARCH: THE TRAVELLING SALESMAN APPROACH

5.1 INTRODUCTION

In this chapter we shall change our objective while keeping the same constraints. That is, we shall leave the bottleneck objective function and focus on minimizing total mission time. This objective function is given by the expression (2.12) and this type of routing problem is called the "Minimum Total Time Routing Problem (MTRP)".

Rather than defining a solution procedure for MTRP, we shall reformulate the problem as classical Travelling Salesman Problem and show that MTRP can be solved by using the TSP approach. The TSP network obtained as a result of this formulation has (2P+M) modes. The formulated TSP network has some specialities which can be utilized while solving the problem. But no special algorithm has been developed for this purpose. Only the well-known solution procedures for TSP are utilized. One can make use of this fact and can improve the TSP procedures to solve MTRP more efficiently.

The MTRP differs from BRP only in objective function. In the case of BRP it is easy to estimate the longest path length (d_{max}), but in case of MTRP it is not so easy to make such an estimate. If it is estimated then the estimate must be a relaxed one, that is, it must be long enough to enable all possibilities, which will certainly blow up the number of paths that should be considered by our previous approach. So we shall change our approach but still utilize essentially the same network transformation.

5.2 A NETWORK TRANSFORMATION FOR THE MINIMUM TOTAL TIME CARGO ROUTING

In order to formulate MTRP as TSP, a network is generated which is similar to the one generated in case of BRP. Here again, some nodes represent the airports which have initially planes and some nodes represent the loads between airports. But artificial source and terminal nodes are omitted. Therefore, the resulting network has (RP+M) nodes. The time matrix related with this network is called V from now on. In this network, nodes 1,...,RP represent the airports which have planes initially, and nodes (RP+1),...,(RP+M) represent loads. Fig. 5.1 shows this formulation on Example A.

Interactions between loads and interactions between the airports and the loads are kept the same. Hence, the meanings and the lengths (i.e. time required to traverse these arcs) of these arcs are the same with bottleneck formulation. Interactions on this network are as follows:

i. Interactions between nodes representing loads.

$$v_{k\ell} = \begin{cases} d_{[L_{(\ell-RP),1}],[L_{(\ell-RP),2}]} + TU + TL, \\ \quad \text{if } L_{(k-RP),2} = L_{(\ell-RP),1}\ ; \\ \\ d_{[L_{(k-RP),2}],[L_{(\ell-RP),1}]} \\ + d_{[L_{(\ell-RP),1}],[L_{(\ell-RP),2}]} + TU + TL, \\ \quad \text{if } L_{(k-RP),2} \neq L_{(\ell-RP),1}\ ; \end{cases} \tag{5.1}$$

$\forall k,\ell,\quad k \neq \ell,\quad \ell,k \in \{RP+1,\ldots,RP+M\}$, where, L, RP, M and d defined in previous chapters.

ii. Interactions between nodes representing airports and nodes representing loads.

$$v_{k\ell} = \begin{cases} d_{[L_{(\ell-RP),1}],[L_{(\ell-RP),2}]} + TU + TL, \\ \quad \text{if } L_{(\ell-RP),1} = [k^{th} \text{ airport which has planes initiall} \\ \\ d_{[k^{th}\text{ airport which has initially planes}],[L_{(\ell-RP),1}]} \\ + d_{[L_{(\ell-RP),1}],[L_{(\ell-RP),2}]} + TU + TL, \\ \quad \text{if } L_{(\ell-RP),1} = [k^{th} \text{ airport which has planes initiall} \end{cases} \tag{5.2}$$

$\forall k,\ell,\quad k = \{1,\ldots,RP\},\ \ \ell \in \{RP+1,\ldots,RP+M\}.$

iii. Interactions between nodes representing loads and nodes representing airports.

Previously no such interaction has been assumed, but in this case we include these arcs and assign them zero time. That is,

$$v_{k\ell} = 0; \quad \forall k,\ell, \quad k \in \{RP+1,\ldots,RP+M\}, \quad \ell \in \{1,\ldots,RP\}. \quad (5.3)$$

iv. Interactions between nodes representing airports.

Again no meaning has been assigned to such interaction previously. But in this case we shall connect some of the airports among each other with arcs of zero time, and not all airports will be connected to each other instead, depending upon our objective we shall select these to be connected.

The arcs which are discussed in interactions (iii) and (iv) have no physical meaning. The reason for their existence will be cleared in following sections.

We shall solve the MTRP over this network by a new approach. In order to introduce this approach we shall now define the classical Travelling Salesman Problem and describe its relation to the vehicle routing problem.

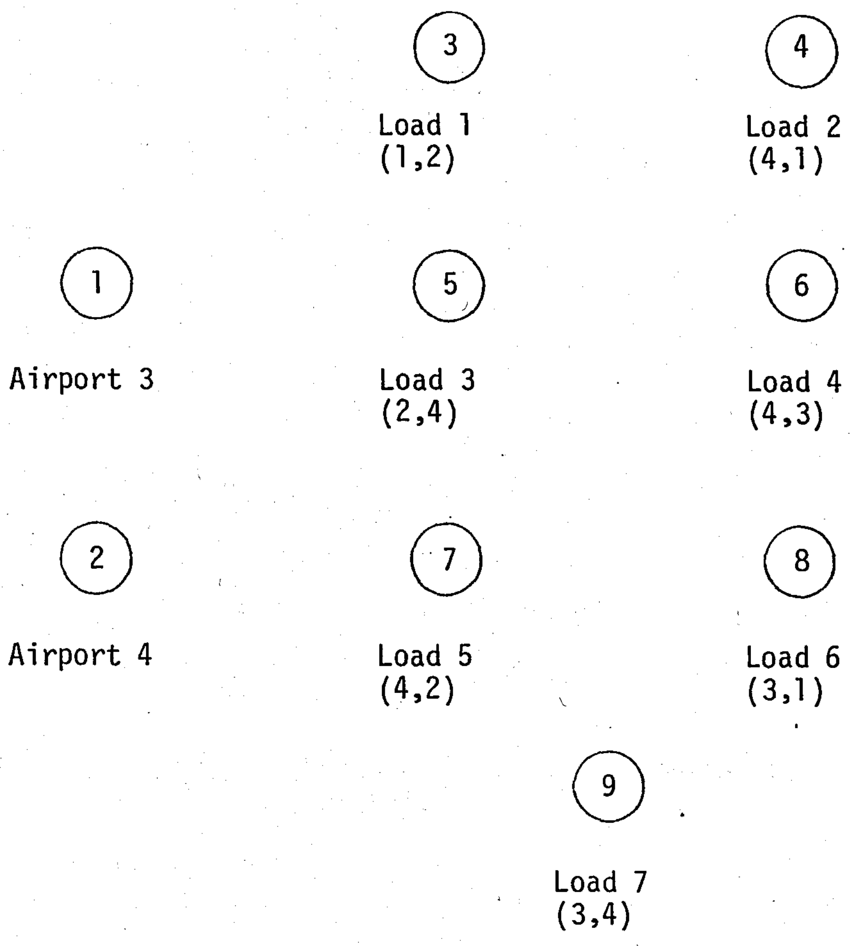


FIGURE 5.1 - The transformed network of Example A

5.3 THE TRAVELLING SALESMAN PROBLEM (TSP) AND ITS EXTENSION OF MULTI-TRAVELLING SALESMAN CASE (MTSP)

The Travelling Salesman problem is a well-known combinatorial problem. It can be defined as follows: Given n cities and a salesman, find the shortest (or least cost) tour such that the salesman visits each city exactly once. That is, he starts from city 1 and visits each of the other (n-1) cities once and only once and then returns to city 1. Thus the problem can be formulated as:

$$\text{Min } Z = \sum_{i=1}^{n} \sum_{J=1}^{n} d_{iJ} x_{iJ} \tag{5.4}$$

$$\sum_{i=1}^{n} x_{iJ} = 1 \quad , \quad \forall J \tag{5.5}$$

$$\sum_{J=1}^{n} x_{iJ} = 1 \quad , \quad \forall i \tag{5.6}$$

$$\underline{X} = (x_{i,J}) \in S \tag{5.7}$$

$$x_{iJ} = \begin{cases} 1 \quad , & \text{if arc } (i,J) \text{ is in the tour;} \\ 0 \quad , & \text{otherwise;} \qquad \forall\, i,J \end{cases} \tag{5.8}$$

d_{iJ} = is the cost of (or length) going from city i to city J; $\forall$ i,J

The set S can be the set of any restrictions to avoid solutions, satisfying constraints (5.5) and (5.6). Such restrictions are called subtour elimination constraints. Generally three definitions for the set S are given in literature (Bodin, Golden and Assad, 1981):

i. $S = \{(x_{iJ}) | \sum_{i \in Q} \sum_{J \notin Q} x_{iJ} \geq 1$ for every non-empty proper subset Q of $[1,\ldots,n]\}$ (5.9)

ii. $S = \{(x_{iJ}) | \sum_{i \in Q} \sum_{J \in Q} x_{iJ} \leq |Q| - 1$ for every non-empty subset Q of $[2,3,\ldots,n]\}$ (5.10)

iii. $S = \{(x_{iJ}) | y_i - y_J + n\, x_{iJ} \leq n-1$ for $2 \leq i \neq J \leq n$ for some real numbers $y_i\}$ (5.11)

The Multiple Travelling Salesman Problem is the generalization of TSP to the case where there are m salesmen instead of one. Initially, all the m salesmen are in one of the cities called "the depot". They will visit some of the cities and will eventually return to the depot. The assignment based formulation of MTSP is a natural extension of TSP formulation.

$$\text{Min } Z = \sum_{i=1}^{n} \sum_{J=1}^{n} d_{iJ} x_{iJ} \tag{5.12}$$

$$\text{s.t. } \sum_{i=1}^{n} x_{iJ} = b_J = \begin{cases} M; & \text{if } J = 1 \text{ (i.e. the depot)} \\ 1; & \text{if } J \neq 1. \end{cases} \tag{5.13}$$

$$\sum_{J=1}^{n} x_{iJ} = a_i = \begin{cases} M; & \text{if } i = 1 \text{ (i.e., the depot)} \\ 1; & \text{if } i \neq 1. \end{cases} \tag{5.14}$$

$$\underline{X} = (x_{iJ}) \in S \tag{5.15}$$

$$x_{iJ} = 0,1 \quad , \quad \forall\ i,J \tag{5.16}$$

Any MTSP problem can be converted into an equivalent TSP. Equivalent TSP formulations of MTSP were derived by Bellmore and

Hong (1974), Svestka and Huckfeldt (1973), Rao (1970), Hong and Padberg (1977), Berenguer (1971) and others. The equivalence is achieved by creating m copies of the depot, each connected to other nodes exactly as the depot is in the original network and by allowing no interactions between the M copies of depots (i.e. arcs between them have assigned infinite lengths). Hence equivalent TSP formulation has (n m-1) dones. As a result of this formulation, an optimal single TSP tour in the enlarged network will never use an arc connecting copies of the depot and this optimal tour can be decomposed into subtours resulting in the optimal solution for MTSP.

For example, in Fig. 5.2.a there are five nodes and two salesmen at node 1. Then, the expanded network will contain nodes D1, D2, 2, 3, 4, 5. Nodes D1 and D2 being the copies of the depot (node 1). Each salesman is assumed to be situated in one of them. In Fig. 5.2.b consider the tour {D1-4-3-D2-2-5-D1} and the interpretation in the two salesmen problem is shown in Fig. 5.2.a. Here the subtours {1-2-5-1} and {1-3-4-1} represent the tours of the individual salesmen.

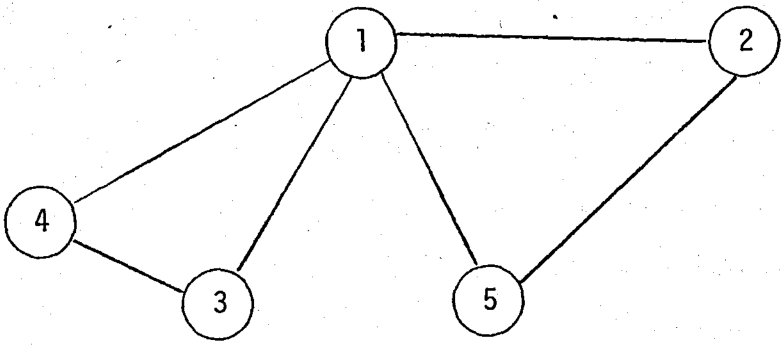


FIGURE 5.2.a - Example of a 5 node, 2 salesmen MTSP tour

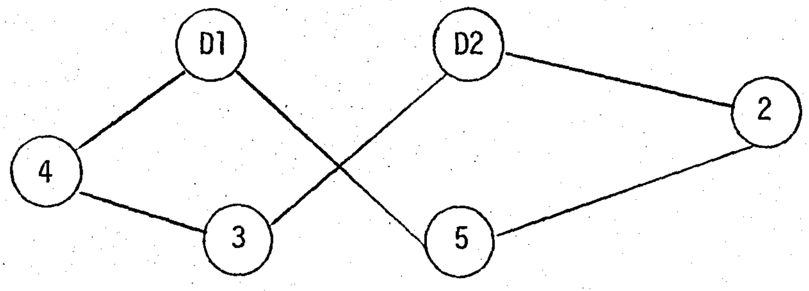


FIGURE 5.2.b - Equivalent TSP tour

5.4 MTSP IN CASE OF MUTLIPLE DEPOTS (MDMTSP)

Multiple TSP can be generalized by assuming the existence of more than one depot. Here again there are n cities to be visited by m salesmen. But salesmen are located at several depots. Let there be d depots and each depot houses prespecified number of salesmen. In the next section we shall show how this problem can be converted to an equivalent TSP. In order to illustrate the meaning of MTSP in case of Multi Depot we shall now define and formulate the Multi Depot Vehicle Routing Problem. One can view MTSP as a special case of Vehicle Routing problem, where vehicles being the salesmen and some of the constraints of VRP are dropped. Also, one can view MDMTSP as a special case of Multi Depot VRP in the same manner.

The Vehicle Routing Problem (VRP), is to obtain a set of delivery routes from a central depot to various demand points, each of which has known requirements, so as to minimize total distance covered by the entire fleet. Vehicles have capacity and maximum route constraints. All vehicles start and finish at the central depot. The mathematical formulation of VRP is given by Golden et al. (1977) as follows,

$$\text{Min } Z = \sum_{i=1}^{n} \sum_{J=1}^{n} \sum_{k=1}^{NV} d_{iJ} x_{iJ}^{k} \tag{5.17}$$

$$\text{s.t.} \sum_{i=1}^{n} \sum_{k=1}^{n} x_{iJ}^{k} = 1 \quad (J = 2,\ldots,n) \tag{5.18}$$

$$\sum_{J=1}^{n} \sum_{k=1}^{NV} x_{iJ}^{k} = 1 \quad (i = 2,\ldots,n) \tag{5.19}$$

$$\sum_{i=1}^{n} x_{ip}^{k} \sum_{J=1}^{n} x_{pJ}^{k} = 0 \quad (k = 1,\ldots,NV; \; p = 1,\ldots,n) \tag{5.20}$$

$$\sum_{i=1}^{n} Q_i \left(\sum_{J=1}^{n} x_{iJ}^{k} \right) \leq P_k \quad (k = 1,\ldots,NV) \tag{5.21}$$

$$\sum_{i=1}^{n} t_i^k \sum_{J=1}^{n} x_{iJ}^{k} + \sum_{i=1}^{n} \sum_{J=1}^{n} t_{iJ}^{k} x_{iJ}^{k} \leq T_k \quad (k = 1,\ldots,NV) \tag{5.22}$$

$$\sum_{J=2}^{n} x_{1J}^{k} \leq 1 \quad (k = 1,\ldots,NV) \tag{5.23}$$

$$\sum_{i=2}^{n} x_{i1} \leq 1 \quad (k = 1,\ldots,NV) \tag{5.24}$$

$$\underline{X} \in S \tag{5.25}$$

$$x_{iJ}^{k} = 0,1 \quad \forall\; i,J,k \tag{5.26}$$

where,

n = number of nodes

NV = number of vehicles

P_k = capacity of vehicle k

T_k = maximum time allowed for a route of vehicle k

Q_i = demand at node i ($Q_1 = 0$, node 1 being the depot)

t_i^k = time required for vehicle k to deliver or collect at node i ($t_1^k = 0$)

t_{iJ}^k = travel time for vehicle k from node i to node J ($t_{ii}^t = \infty$)

d_{iJ} = shortest distance from node i to node J.

$$x_{iJ}^k = \begin{cases} 1, & \text{if arc } (i,J) \text{ is traversed by vehicle } k \\ 0, & \text{otherwise} \end{cases}$$

$\underline{X}$ = matrix with components $x_{iJ} = \sum_{k=1}^{NV} x_{iJ}^k$, specifying connections regardless of vehicle type.

Equations (5.18) and (5.19) ensure that each demand node served by exactly one vehicle. Equations (5.20) represent route continuity, that is if a vehicle enters to a demand node then it must exit from that node. Equations (5.21) are the vehicle capacity constraints and Equations (5.22) are the total elapsed time constraints. Equations (5.23 and (5.24) ensures that the vehicle availability is not exceeded. Finally Equations (5.25) are the subtour elimination constraints.

The mathematical programming formulation of vehicle routing problem is altered in a minor way to incorporate multiple depots. Let nodes 1,2,...,M denote the depots. We obtain the formulation of Multi-Depot Vehicle Routing by changing the index in constraints (5.18) and (5.19) to (J = M+1,...,n) and by changing constraints (5.23) and (5.24) as follows:

$$\sum_{i=1}^{M} \sum_{J=M\ 1}^{n} x_{iJ}^{k} \leq 1 \qquad (k = 1,\ldots,NV) \qquad (5.27)$$

$$\sum_{p=1}^{M} \sum_{i=M\ 1}^{n} x_{ip}^{k} \leq 1 \qquad (k = 1,\ldots,NV) \qquad (5.28)$$

But in multi-depot case we must redefine choices for subtour elimination constraints as follows,

i. $S = \{(x_{iJ}) \mid \sum_{J\varepsilon Q} \sum_{J\not\varepsilon Q} x_{iJ} \geq 1$ for every proper subset Q of V containing nodes $[1,2,\ldots,M]\}$ (5.29)

ii. $S = \{(x_{iJ}) \mid \sum_{i\varepsilon Q} \sum_{J\varepsilon Q} x_{iJ} \leq |Q| - 1$ for every non-empty subset Q of $\{M+1,\ M+2,\ldots,n\}\}$ (5.30)

iii. $S = \{(x_{iJ}) \mid y_i - y_J + n\, x_{iJ} \leq n - 1$ for $M+1 \leq i \neq J \leq n$ for some real numbers $y_i\}$ (5.31)

If we drop the capacity (5.21) and elapsed time (5.22) constraints from the formulations of VRP and Multi-Depot VRP, then we can obtain the mathematical programming formulation of MTSP and MDMTSP respectively.

Note that, the initial depots of vehicles are irrelevant in this formulation, also whether the vehicles to their initial depot or not, is not controlled by this formulation. The only requirement is the utilization of at most the given number of vehicles. These points will be discussed in the next sections.

5.5 THE TRANSFORMATION OF MDMTSP TO AN EQUIVALENT TSP

5.5.1 Introduction

In this section we shall illustrate how a given MDMTSP is converted into an equivalent TSP. This transformation in principle similar to that of utilized in MTSP network. After this transformation an asymmetric TSP network is obtained with (2m+n) dones where n is the number of cities and m is the number of salesmen.

The transformation is realized by generating duplicates of the depots. In the case of MTSP for each salesmen one copy of the depot is generated. But in this case we utilize two copies of the related depot for each salesman. The duplicates of the depots are called "dummy nodes" from now on and there are 2m dummy nodes. Each salesman will begin its tour from one of the dummy nodes and finish it at another dummy node. The first dummy node is called the "departure node" and the second one is called the "arrival node" for that salesman. The network of MDMTSP is transformed so that, there are no arcs entering to the departure nodes, but just arcs leaving. Indeed these arcs are the same arcs leaving the depot on the original MDMTSP network. Similarly, just the converse is true for arrival nodes.

This way, the salesman will leave the depot from the arrival node. So he will make a subtour starting and ending at the depot. In order to enable other salesman tours, an arc put from each arrival node to next departure node with zero length which directs the TSP tour to the next departure. Thus, the TSP tour is forced to cover arcs between dummy nodes, since there is no arc leaving the arrival nodes except the ones that are connected to departure nodes. Since these arcs have zero length, they do not change the objective function of solution obtained from the TSP tour.

5.5.2 An Example Transformation

Consider the network given in Fig. 5.3.a. There are 10 cities and 2 depots, D1 and D2. There are 2 salesmen at depot D1 (called salesman A and salesman B) and 1 salesman at depot D2 (called salesman C). The distance matrix of this network is given at Table 5.1.a.

The equivalent TSP network is shown in Fig. 5.3.b and related distance matrix is given at Table 5.1.b.

The dummy nodes are interpreted as follows:

Node 11: Departure node of Salesman A at Depot 1.

Node 12: Arrival node of Salesman A at Depot 1.

Node 13: Departure node of Salesman B at Depot 1.

Node 14: Arrival node of Salesman B at Depot 1.

Node 15: Departure node of Salesman C at Depot 2.

Node 16: Arrival node of Salesman C at Depot 2.

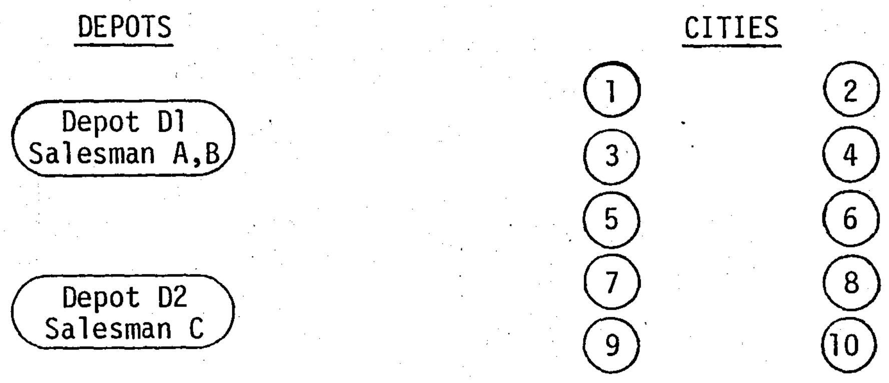


FIGURE 5.3.a - Original network of Example 5.5.2

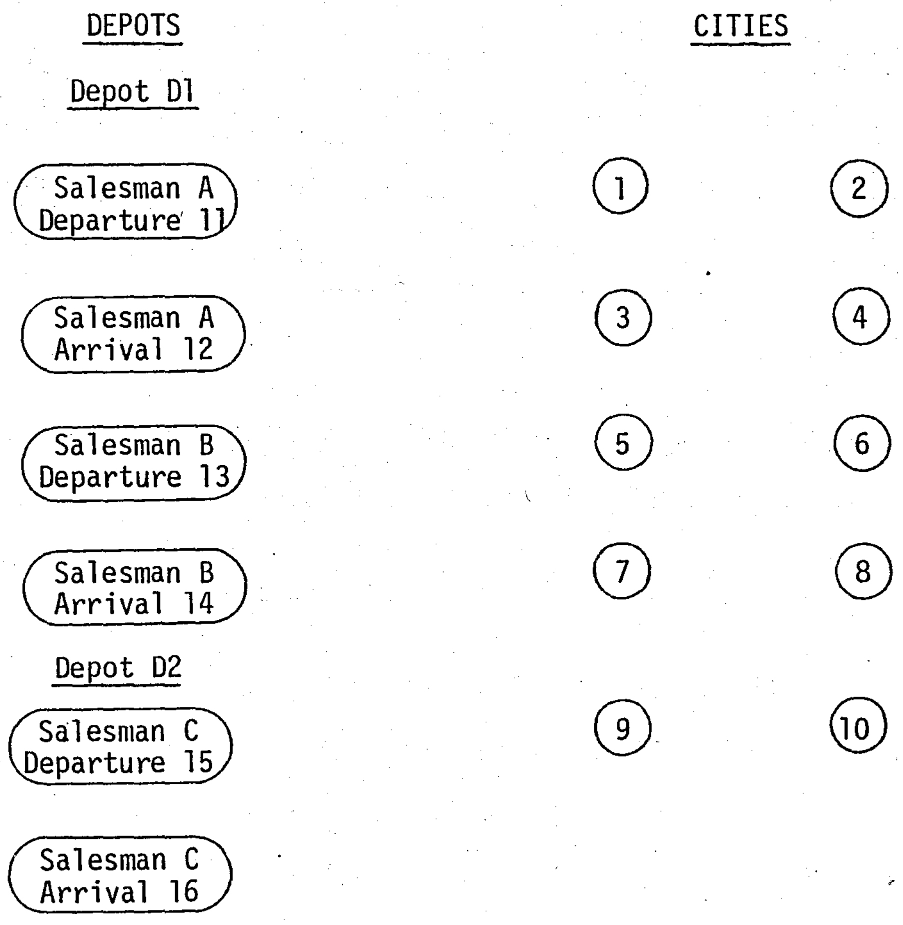


FIGURE 5.3.b - Equivalent TSP network of Example 5.5.2

TABLE 5.1.a - Original Distance Matrix of Example 5.5.2

	D1	D2	1	2	3	4	5	6	7	8	9	10
D1	∞	∞	3	4	2	∞	∞	∞	∞	∞	∞	∞
D2	∞	∞	∞	∞	7	∞	5	∞	8	∞	∞	∞
1	3	∞	∞	20	5	10	5	8	7	3	2	1
2	4	∞	20	∞	6	9	12	16	5	13	18	13
3	2	7	5	6	∞	11	19	17	7	9	10	8
4	∞	∞	10	9	11	∞	14	20	15	7	6	8
5	∞	5	5	12	19	14	∞	8	10	13	9	7
6	∞	∞	8	16	17	20	8	∞	5	8	16	17
7	∞	8	7	5	7	15	10	5	∞	6	12	14
8	∞	∞	3	13	9	7	13	8	6	∞	15	13
9	∞	∞	2	18	10	6	9	16	12	15	∞	9
10	∞	∞	1	13	8	8	7	17	14	13	9	∞

Now, let us consider the following tours in TSP network and try to construct the related subtours in MDMTSP network.

Tour 1: Let a TSP tour be

{11-2-4-1-12-13-3-14-15-5-6-8-10-9-7-16-11}.

Then the corresponding subtours in MDMTSP network are,

Salesman A: D1-2-4-1-D1

Salesman B: D1-3-D1

Salesman C: D2-5-6-8-10-9-7-D2.

TABLE 5.1.b - Equivalent Distance Matrix of Example 5.5.2

	A	A	B	B	C	C										
	Dep	Arr	Dep	Arr	Dep	Arr	1	2	3	4	5	6	7	8	9	10
A Dep	∞	∞	∞	∞	∞	∞	3	4	2	∞	∞	∞	∞	∞	∞	∞
A Arr	∞	∞	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
B Dep	∞	∞	∞	∞	∞	∞	3	4	2	∞	∞	∞	∞	∞	∞	∞
B Arr	∞	∞	∞	∞	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
C Dep	∞	∞	∞	∞	∞	∞	7	∞	∞	∞	5	∞	8	∞	∞	∞
C Arr	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	∞	3	∞	3	∞	∞		20	5	10	5	8	7	3	2	1
2	∞	4	∞	4	∞	∞	20	∞	6	9	12	16	5	13	18	13
3	∞	2	∞	2	∞	7	5	6	∞	11	19	17	7	9	10	8
4	∞	∞	∞	∞	∞	∞	10	9	11	∞	14	20	15	7	6	8
5	∞	∞	∞	∞	∞	5	5	12	19	14	∞	8	10	13	7	7
6	∞	∞	∞	∞	∞	∞	8	16	17	20	8	∞	5	8	16	17
7	∞	∞	∞	∞	∞	8	7	5	7	15	10	5	∞	6	12	14
8	∞	∞	∞	∞	∞	∞	3	13	9	7	13	8	6	∞	15	13
9	∞	∞	∞	∞	∞	∞	2	18	10	6	9	16	12	15	∞	9
10	∞	∞	∞	∞	∞	∞	1	13	8	8	7	17	14	13	9	∞

Tour 2: Let a TSP tour be,

{11-2-14-15-3-12-13-1-4-5-6-8-10-9-7-16-11}.

Then the corresponding subtours in MDMTSP network are,

Salesman A: D1-2-D1

Salesman B: D2-3-D1

Salesman C: D1-1-4-5-6-8-10-9-7-D2.

5.5.3 Analysis of Transformation Suggested

As indicated in the previous section, the initial depot of salesman is always controllable in this suggested network transformation. But we have two possibilities for final depot. They are;

i. The initial and final depots of salesmen are the same, what is expected.

ii. The initial and final depots of salesman can be different, which may be desirable or not depending upon the particular application. This transformation enables such tours, but in case of routing planes, it will be shown that there is no violating effect.

In Section 5.5.1 it has been suggested that each arrival node should be connected to the next departure node. Without loss of generality we can restate this transformation as follows: each arrival node should be connected to one or more arrival nodes by an arc of zero length.

5.5.4 Number of Salesmen Utilized in MDMSTP

If we connect the departure node of a salesman to its arrival node by an arc of zero length, then we will create for the TSP tour the possibility of not utilizing that salesman at all.

A solution which does not utilize some of the salesman can be explained as follows;

i. There are alternative optimal solutions which utilize different number of salesmen.

ii. Utilizing extra salesmen may effect objective function in two ways. If it decreases the objective function value, then TSP algorithm automatically selects that solution. The interesting case is the next one in which utilizing extra salesman may increase the objective function value although extra salesmen do not incur any cost. The major reason for this is the alteration of triangular inequality in the distance matrix. For exmaple, in Fig. 5.4, such a case for a single depot with two salesmen is shown. In this case optimal solution with one salesman will be the tour {D-3-4-2-1-D} with the objective function value 35. Any solution which utilizes more than one salesman will generate higher objective function values.

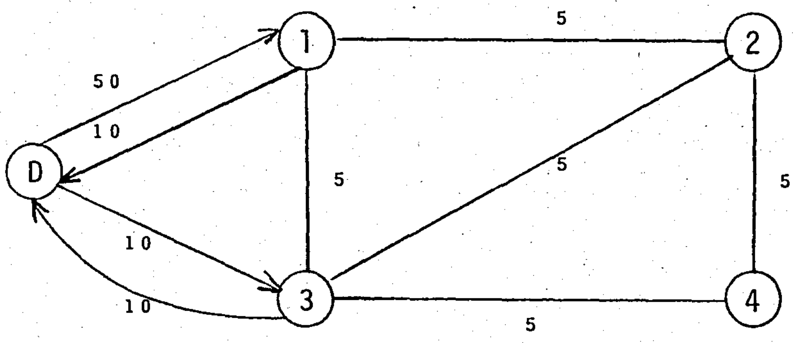


FIGURE 5.4 - Example of a single depot two salesman problem

5.6 THE MULTI DEPOT MULTI SALESMEN TSP WITH FIXED COSTS

5.6.1 <u>The Single Depot Case</u>

Although in some applications, exactly m salesmen are required, there are cases where there is a cost of each salesman and it is desirable to utilize as few as possible salesmen due to that cost. That is, using r salesmen (where $1 \leq r \leq m$).

Hong and Padberg (1977) define this problem as: "By assigning the i^{th} salesman to a tour, one incurs a fixed charge f_i, which is independent of his tour. For travelling from city i to city J, one incurs a cost c_{iJ} that does not depend upon which salesman makes that particular trip. The problem is to find the number of salesmen to be employed and their respective routes so as to minimize total cost."

The MTSP subject to fixed charges is abbreviated as MTSPF. Bellmore and Hong (1974) and Rao (1980) and Discenza (1981) have shown possible transformations to include fixed costs to the MTSP formulation.

Bellmore and Hong (1974) have shown that asymmetric MTSPF in (n+1) cities, one being the depot, with m salesmen is equivalent to standard asymmetric TSP on (n+m-1) cities. Their transformation is as follows:

For the sake of simplicity in notation, note that the depot node is called node 0 and the n remaining cities are to be visited by m salesmen.

Let nodes labelled -1, -2, ..., -(m-1) denote the additional nodes put to convert an MTSP to a TSP.

Therefore nodes 0, -1,...,-(m-1) represent the copies of the original depot.

Let D be the original distance matrix and D' be the expanded distance matrix. Then, the element of D', d'_{iJ} are expressed in terms d_{iJ} and f_i as follows:

$$d'(i,J) = d(i,J) \qquad i = 1,\ldots,n \;;\; J = 1,\ldots,n \qquad (5.32)$$

$$\begin{aligned} d'(-i,J) &= d(0,J) + 0.5f_i & i &= 0,1,\ldots,(m-1) \\ d'(J,-i) &= d(J,0) + 0.5f_i & J &= 1,2,\ldots,n \end{aligned} \qquad (5.33)$$

$$d'(-i,-(i-1)) = 0.5f_{i-1} - 0.5f_i \qquad i = 1,2,\ldots,(m-1) \quad (5.34)$$

Figure 5.5 shows an example of this transformation on a 5 cities, 3 salesmen problem. Numbers on arcs represent distance of that arc. Now, consider the tour (0,1,4,-2,-1,2,3,0). f_0 is added as one half along arc (0,1) and as one half along (3,0). $0.5f_1$ are added along (-2,-1) and (-1,2). $0.5f_2$ are added along (4,-2) and subtracted along (-2,-1), thus cancelling each other.

This tour can be interpreted as:

Salesman 0 visits cities 1 and 4.

Salesman 1 does not visit any city.

Salesman 2 visits cities 2 and 3.

Hong and Padberg (1977) have shown that a symmetric MTSPF can be transformed to a standard TSP by using (n+m+4) nodes. Later Rao (1980) has proven that this can be done by using only (n+m-1) nodes.

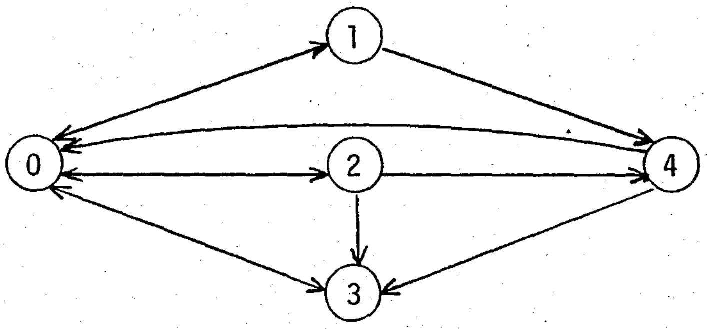


Fig. 5.5.a - An example of MTSPF on 5 cities

Following theorems will prove that these transformations are valid.

THEOREM 5.1: (Rao, 1980) For every r tour on MTSPF network for $1 \leq r \leq m$, there is a tour on equivalent TSP network satisfying,

$$Z = \sum_{k=1}^{r} Z_k + \sum_{k=0}^{r-1} f_k$$

where Z is the distance matrix of the tour on TSP network, Z_k is the distance of the kth cycle in the r-tour on MTSPF network and f_k is the cost associated with some salesman k.

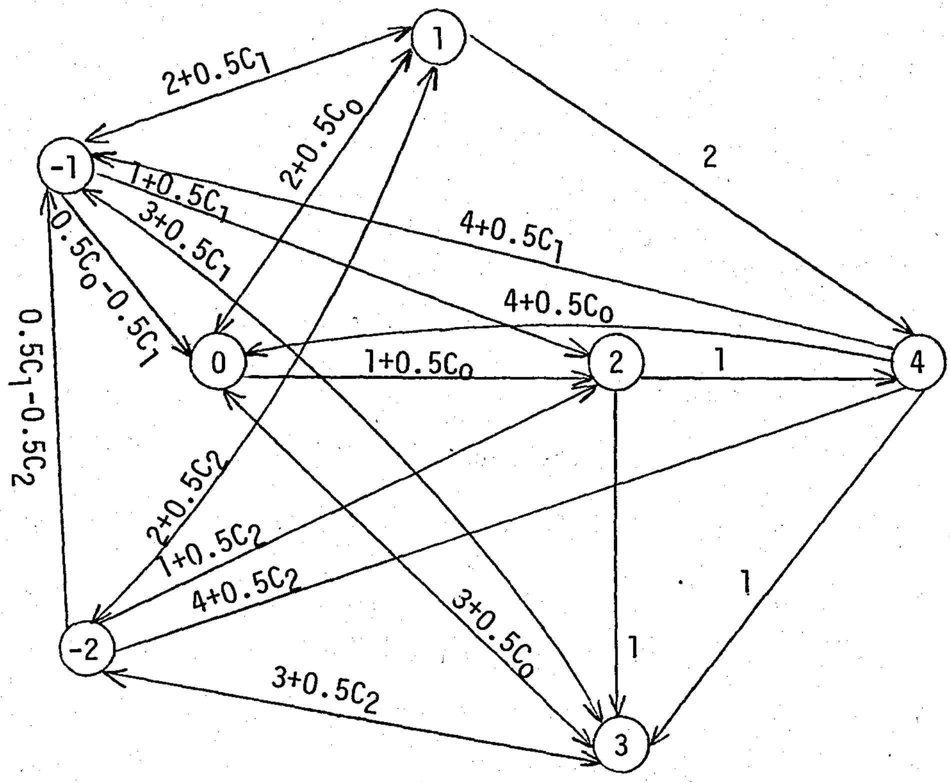


FIGURE 5.5.b - The equivalent TSP formulation

THEOREM 5.2: (Rao, 1980) For every tour on TSP with distance Z, there is an r-tour (utilizing r salesmen) on equivalent MTSPF network for some r ($1 \leq r \leq m$) such that,

$$Z \leq \sum_{k=1}^{r} Z_k + \sum_{k=o}^{r-1} f_k$$

where Z_k is the distance matrix of the kth cycle in r-tour, and $f_o \leq f_1 \leq f_2 \leq \dots \leq f_{m-1}$.

The proofs are easy. They are essentially proved by tracing salesmen in both the MTSPF network and the equivalent TSP network.

5.6.2 The Multiple Depot Case

The ideas developed for the single depot case can be applied to multiple-depot case if there exists fixed charges of utilizing salesman. Reader should note here that the distance matrix of the MDMTSP network should be compatible with fixed costs of the salesman. Then, introduction of fixed costs can be accomplished by adding fixed cost f_i of utilizing salesman i to all arcs leaving from departure node of salesman i. But in this case the dummy arcs defined in Section 5.5.4 should be put in the network so to enable not utilizing salesman i.

If salesman i is utilized, then the solution should cover exactly one of the arcs leaving from the related departure node, thus fixed cost of utilizing salesman i is added to the objective function value. On the other hand, if salesman i is not utilized, then this means the arc between departure and arrival nodes of salesman i is covered by that solution and no fixed cost related with this salesman incurred.

Let us consider the same network as in Section 5.5.2. Assume that salesmen A,B,C have fixed costs f_a, f_b, f_c respectively, and let their values are 10, 20, 30. The cost matrix obtained as a result of this transformation is given at Table 5.2.

TABLE 5.2 - The Cost Matrix of Example 5.5.2 With Fixed Costs

	A Dep	A Arr	B Dep	B Arr	C Dep	C Arr	1	2	3	4	5	6	7	8	9	10
A Dep	∞	0	∞	∞	∞	∞	13	14	13	∞	∞	∞	∞	∞	∞	∞
A Arr	∞	∞	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
B Dep	∞	∞	∞	0	∞	∞	23	24	22	∞	∞	∞	∞	∞	∞	∞
B Arr	∞	∞	∞	∞	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
C Dep	∞	∞	∞	∞	∞	0	∞	∞	37	∞	35	∞	38	∞	∞	∞
C Arr	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	∞	3	∞	3	∞	∞	∞	20	5	10	5	8	7	3	2	1
2	∞	4	∞	4	∞	∞	20	∞	6	9	12	16	5	13	18	13
3	∞	2	∞	2	∞	7	5	6	∞	11	19	17	7	9	10	8
4	∞	∞	∞	∞	∞	∞	10	9	11	∞	14	20	15	7	6	8
5	∞	∞	∞	∞	∞	5	5	12	19	14	∞	8	10	13	9	7
6	∞	∞	∞	∞	∞	∞	8	16	17	20	8	∞	5	8	16	17
7	∞	∞	∞	∞	∞	8	7	5	7	15	10	5	∞	6	12	14
8	∞	∞	∞	∞	∞	∞	3	13	9	7	13	8	6	∞	15	13
9	∞	∞	∞	∞	∞	∞	2	18	10	6	9	16	12	15	∞	9
10	∞	∞	∞	∞	∞	∞	1	13	8	8	7	17	14	13	9	∞

Now, consider the tour in this network,

{11-12-13-1-2-4-3-14-15-5-6-7-10-9-8-16-11}

The interpretation of this tour on original network is, Salesman A: not utilized,

Salesman B: D1-1-2-4-3-D1,

Salesman C: D2-5-6-7-10-9-8-D2.

Since non of the arcs (11-1), (11-2) and (11-3) is covered, the fixed cost of Salesman A is not incurred. Fixed cost of Salesman B is incurred on arc (13-1) and fixed cost of Salesman C is incurred on arc (15,5).

5.7 THE FORMULATION OF MINIMUM TOTAL TIME ROUTING AS AN MDMTSP

After discussing how MDMTSP can be solved, we can formulate Minimum Total Time Routing Problem as an MDMTSP. In this section.we shall utilize the network developed in Section 5.1 as the original network of MDMTSP. Within that network, assume that nodes representing airports which have planes initially as depots, nodes representing loads as cities, and planes as salesmen. Thus we have RP depots, M cities and P planes available. Therefore after transformation to TSP network we shall have (2P+M) nodes.

Fig. 5.6 shows the MDMTSP network of Example A. After the transformation to TSP we shall obtain the time matrix (it was referred to as the distance matrix in MDMTSP transformation, but in this case, that matrix correspond to the operation time matrix defined in Chapter 3) given at Table 5.3.

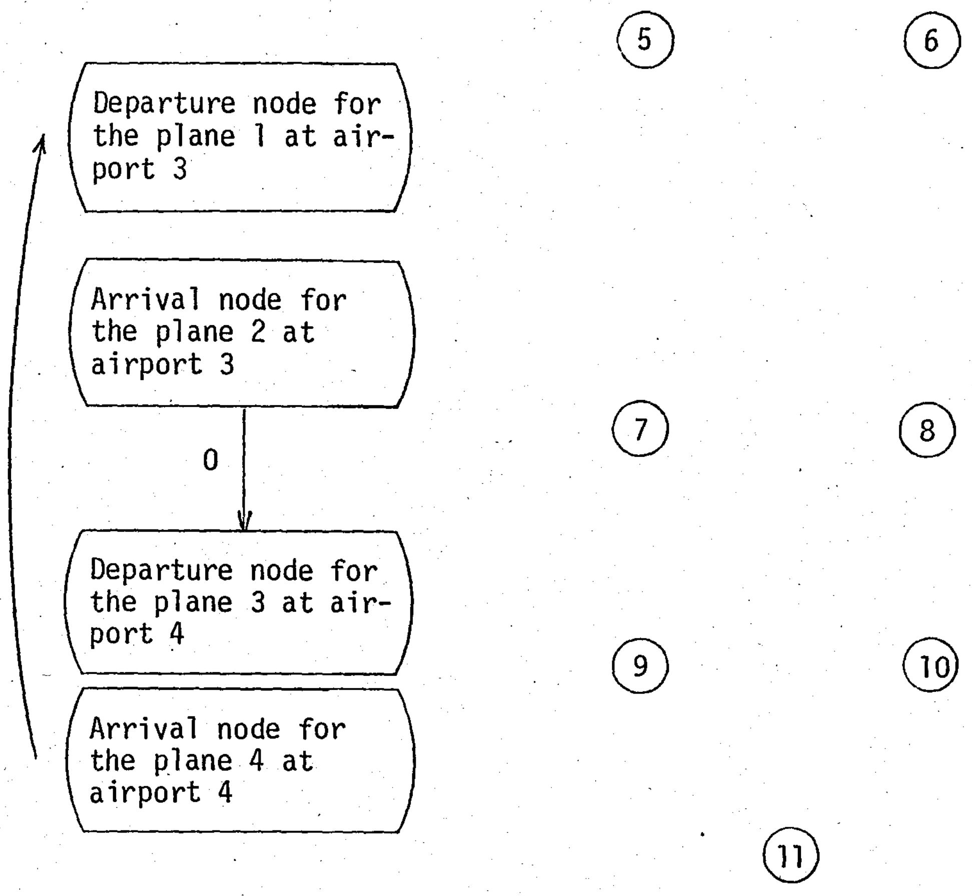

Departure node for the plane 1 at airport 3
Arrival node for the plane 2 at airport 3
0
Departure node for the plane 3 at airport 4
Arrival node for the plane 4 at airport 4
5
6
7
8
9
10
11

FIGURE 5.6 - MDMTSP formulation of Example A

TABLE 5.3 - The Equivalent TSP Matrix of Example A

	1	2	3	4	5	6	7	8	9	10	11
1	∞	∞	∞	∞	110	75	80	55	70	80	35
2	∞	∞	0	∞	∞	∞	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	85	55	85	35	50	100	55
4	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
5	∞	0	∞	0	∞	90	50	70	85	110	65
6	∞	0	∞	0	45	∞	80	75	90	145	100
7	∞	0	∞	0	85	55	∞	35	50	100	55
8	∞	0	∞	0	110	75	80	∞	70	80	35
9	∞	0	∞	0	75	90	50	70	∞	110	65
10	∞	0	∞	0	45	95	80	75	90	∞	100
11	∞	0	∞	0	85	55	85	35	50	100	∞

5.8 EVALUATION

Our aim in this chapter is to suggest a formulation to minimum total time cargo routing problem and to suggest a transformation on MDMTSP network to obtain an equivalent TSP network. There are heuristic and exact methods available to solve TSP. Little et al. (1963), Held and Karp (1970), Miliotis (1976), Crowder and Padberg (1980) suggested exact procedures whereas Rosentkrantz et al. (1977), Clarke and Wright (1964), Norback and Love (1977), Kim (1975), Christofides (1976), Lin (1965) have suggested heuristic procedures. However, in our case the final TSP matrices have some properties which can be utilized in the solution of TSP. It is certain that all TSP tours will cover arcs defined between arrival and departure nodes. Then there is no need to carry on an optimization process on these arcs. If one resorts to methods available for TSP, then one has to make unnecessary calculation on such arcs. Several authors (Russel, 1977) have worked on such TSP cost matrices and suggested techniques to solve them. But still there is need to define a more powerful solution algorithm for the TSP matrix defined in this chapter.

Since we have expressed MTRP as a MDMTSP all arguments about MDMTSP hold. That is, we can connect the departure of a plane to its arrival, so as to create the option of not utilizing that plane at all. This fact is applied to previous problem. Although it utilized again two planes, the load assignments to planes have changed. Thus we caught an alternative optimal solution to the problem.

We can also assign fixed costs to planes and add these costs to appropriate arcs, if we define operation time matrix in terms of monetary units.

Since in case of MTRP we do not care whether the planes are returning to their initial depots or not, interactions between nodes representing loads and nodes representing airports are put in this formulation in order to build up a complete network without affecting the value of the objective function.

VI. CONCLUSIONS AND EXTENSIONS OF RESEARCH

6.1 REAL LIFE CASES AND CONCLUSIONS

The relative effectiveness of the algorithms and formulations developed within this thesis are tested on the sample problems given in Appendix A. These problems are designed to illustrate the principles of the suggested methods. But in this section we shall focus on some realistic cases and discuss their characteristics.

Most common feature of a regional war is that it will take place through some boundary of the country at hand and the material should be transferred to the locations of action from other parts of the country. In such situations, for a particular item some of the points serve as demand points and others as source points; and most of the time the direction of the material flow is between source and demand points. The source points are the bases of the airforce throughout the country and demand points are either the airports in the vicinity of action area or open areas on which material can be landed by some means.

We shall analyze the behaviour of the suggested techniques when such a flow pattern exists and try to see the responses when this strict flow pattern is altered in several ways on some example problems throughout this section. While applying these ideas we shall try to solve another strategical problem, which is where to allocate available planes initially.

For each of the example problems we shall discuss, the difficulties appearing while applying the steps of the BRP Algorithm given in Chapter 3, namely the Path Generation, Elimination, and Search Steps. Before getting in depth analysis of the results obtained it would be better to make some definitions.

Note that the maximum number of paths that can be generated with load cardinality r is limited. Let us have M loads. Then these M loads can be permuted at most ${}_M P_r$ ways. That is, there can be at most ${}_M P_r$ paths generated with load cardinality r, where,

$$ {}_M P_r = \frac{M!}{(M - r)!} \tag{6.1}$$

Also, one could generate at most ${}_M \overline{P}_R$ paths with load cardinality less than or equal to R, where,

$$ {}_M \overline{P}_R = \sum_{r=1}^{R} {}_M P_r \tag{6.2}$$

Similar results are obtained when we consider the maximum number of paths after the elimination process. Since the problem is then: In how many ways one can select r loads among M loads? This time the number of possible combinations ${}_M C_r$ is the maximum number of possible paths with load cardinality r, where,

$$ {}_M C_r = \frac{M!}{r!(M - r)!} \tag{6.3}$$

Then, the $\overline{M}^{C}R$ is the maximum number of paths after elimination process with load cardinality less than or equal to R. The combination and permutation value of some critical M and r values are shown in Appendix B.

If there exists more than one airport (say RP) which have planes initially, then the above figures should be mutliplied by the number of available airports, in order to get the number of possible paths.

Now, it would be better to define the following in order to avoid repetitions later in the text.

P_1 : is the number of paths generated.

P_2 : is the number of paths after elimination.

LC_{max}: is the maximum load cardinality obtained among generated paths

P_{max} : is the maximum number of paths that can be obtained after the elimination process, and can be expressed as follows:

$$P_{max}: RP \times (\overline{M}^{C}L\ C_{max}) \tag{6.4}$$

P_3 : is the number of paths after the elimination process whose time requirement is less than or equal to optimal value of the given BRP.

The test network designed to apply above conditions has 6 airports with 3 of them being demand points and rest of them being supply points. It is assumed that there are only 4 planes available.

The flight times between these airports given in Table 6.1 and time of loading and unloading are taken as 10 and 5 minutes respectively.

TABLE 6.1 - The Flight Time Data of the 6 Airport Problem

	1	2	3	4	5	6	7
1	-	20	45	55	75	90	90
2	20	-	25	45	75	80	50
3	45	25	-	65	95	95	55
4	55	45	65	-	40	35	30
5	75	75	95	40	-	40	60
6	90	80	95	35	40	-	40
7	70	50	55	30	60	40	-

EXAMPLE 6.1: This first problem is designed to illustrate the strict flow pattern. That is, all the 12 loads have to be carried from source points to demand points. In Fig. 6.1 these loads are drawn and it is assumed that the action takes place in the vicinity of airports 1, 2, and 3. The airports 5 and 6 are assumed to be the bases with some planes initially and airport 4 is a base with no planes initially. Note that in order to get quick results in the test runs, the total number of available planes is limited rather than limiting the number of planes at each airport.

Assuming the maximum job time is 490 minutes, the paths generated have following characteristics.

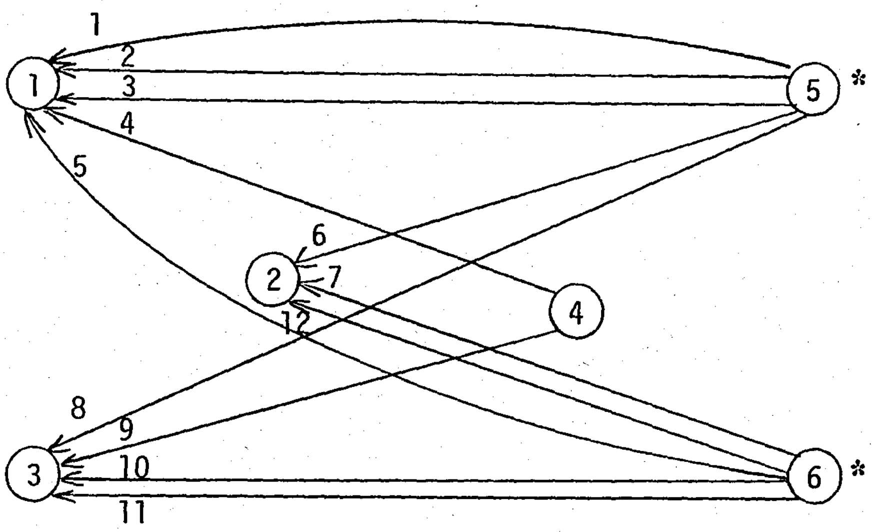


FIGURE 6.1 - The flow pattern given in Example 6.1

$P_1 = 2168$ $P_2 = 572$ $LC_{max} = 3$

note, $RP = 2$, $M = 12$, and $P = 4$

The maximum number of possible paths with load cardinality less than or equal to 3 is 596 ($P_{max} = 596$). Hence the search procedure should scan nearly all the possible combinations. The alternative optimal solutions of this problem are obtained as follows.

SOLUTION 1:

Plane	Initial airport	Loads carried	Time (in minutes)
1	6	7 - 12 - 3	435
2	6	11 - 4 - 8	430
3	6	5 - 1 - 2	435
4	6	10 - 6 - 9	420
		TOTAL:	1720

SOLUTION 2:

Plane	Initial airport	Loads carried	Time (in minutes)
1	6	7 - 6 - 12	435
2	6	10 - 4 - 8	430
3	6	5 - 1 - 2	435
4	6	11 - 3 - 9	430
		TOTAL:	1730

SOLUTION 3:

Plane	Initial airport	Loads carried	Time (in minutes)
1	6	7 - 6 - 12	435
2	6	10 - 4 - 8	430
3	6	5 - 1 - 2	435
4	6	3 - 9 - 11	435
		TOTAL:	1730

Note $z^* = 435$ minutes and $P_3 = 352$ which means that even if we do know the optimal value of the problem and seek the optimal solution, we still have to scan 59.06% of all possible combinations. This analysis illustrates us how the computational effort will grow up as the problem size gets larger. This phenomenon is the most common feature of NP hard class of problems. In BRP problems a factorial function namely P_{max} governs the computational effort that has to be spent. The relationship between problem size and P_{max} will be analyzed later in this section.

EXAMPLE 6.2: This example is designed to alter the flow pattern given in Example 6.1 by introducing two loads, one between source points and the other between demand points. The resulting loads shown in Fig. 6.2.

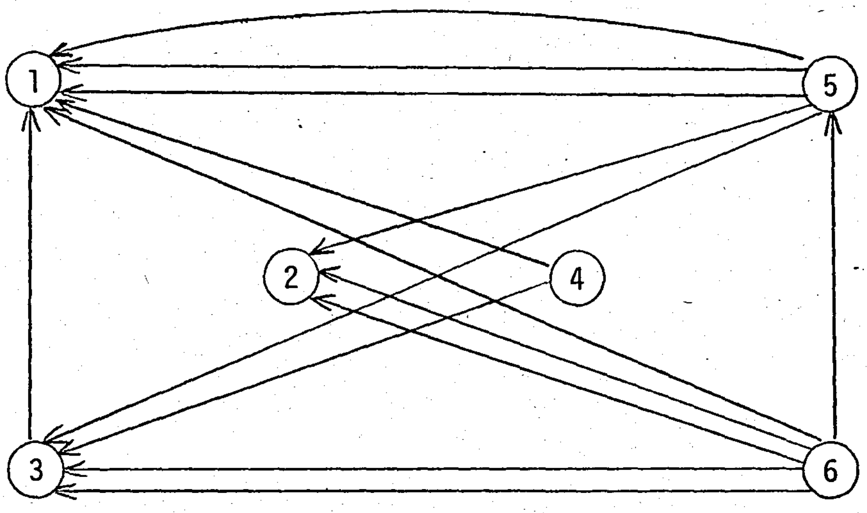


FIGURE 6.2 - The flow pattern given in Example 6.2

Using the same initial airports and maximum time constraints as in Example 6.1, the generated paths resulted in

$P_1 = 6388$ $P_2 = 1361$ $LC_{max} = 5$ and the alternative optimal solutions are obtained as follows;

SOLUTION 1:

Plane	Initial airport	Load carried	Time (in minutes)
1	6	12 - 2 - 10	460
2	6	5 - 1 - 8	455
3	6	7 - 9 - 13 - 3	445
4	5	6 - 4 - 14 - 11	445
			TOTAL: 1805

SOLUTION 2:

Plane	Initial airport	Loads carried	Time (in minutes)
1	5	2 - 4 - 8 - 13	460
2	5	1 - 14 - 5 - 9	460
3	6	7 - 3 - 10	460
4	5	6 -12 - 11	455
		TOTAL:	1835

SOLUTION 3:

Plane	Initial airport	Loads carried	Time (in minutes)
1	5	2 - 4 - 8 - 13	460
2	5	1 -14 - 5 - 9	460
3	6	7 - 3 - 10	460
4	5	2 - 6 - 11	450
		TOTAL:	1830

SOLUTION 4:

Plane	Initial airport	Loads carried	Time (in minutes)
1	6	5 - 6 - 9 - 13	455
2	5	14 -12 - 4 - 8	450
3	6	11 - 1 - 3	460
4	6	7 - 2 - 10	460
		TOTAL:	1825

SOLUTION 5:

Plane	Initial airport	Loads carried	Time (in minutes)
1	6	12 - 1 - 8	445
2	5	2 - 4 -14 - 5	450
3	6	7 - 3 - 11	460
4	6	10 -13 - 6 - 9	460
		TOTAL:	1815

SOLUTION 6:

Plane	Initial airport	Loads carried	Time (in minutes)
1	5	14 - 7 - 4 - 9 -13	460
2	6	12 - 2 -10	460
3	6	5 - 3 - 8	455
4	5	1 - 6 -11	445
			TOTAL: 1820

Note that z* = 460 minutes and P_3 = 1073, although the number of loads increased by less than 15%. The number of paths that has to be considered during search phase is tripled. The major reason for this explosion is the arbitrary configuration of loads 13 and 14.

EXAMPLE 6.3: This example is designed to see the effect of the existence of counter loads, that is, the loads from demand points to source points. To achieve this, load 15 is added between airports 1 and 5 and all other aspects of the problem are kept the same. The paths generated in this fashion yield the following results:

$$P_1 = 12719 \qquad P_2 = 2437 \qquad LC_{max} = 5$$

and the optimal solution is obtained as follows:

Plane	Initial airport	Loads carried	Time (in minutes)
1	6	12 - 2 - 10	460
2	6	5 - 1 - 8	455
3	6	7 - 9 - 13 -15 - 3	460
4	5	6 - 4 - 14 - 11	445
			TOTAL: 1820

Note z* = 460 minutes and P_3 = 1909. The optimization on this data is done on purpose to get the path requirements and one counter load nearly doubled the number of paths that has to be considered. Indeed there is no need to carry on optimization on this data if we know the optimal solution of the previous problem. The optimal solution of the Example 6.2 resulted in several empty flights between airports 1 and 5 so as to carry loads between 5 and 1, as expected. Note that, in the first alternative solution of the previous problem, plane 3 has an empty flight between 1 and 5, and has 15 minutes slack time which is equal to loading plus the unloading time of a single load. So, this plane can cover a load between these airports without violating the optimal value of the problem.

This discussion directs us some interesting results. If there exists some counter loads, then these counter loads will immediately blow up the number of paths that has to be considered. In case of their existence, the best thing to do is drop them first and solve the reduced problem and then try to build up a solution to the original problem utilizing the minimum total cost alternative solution of the reduced problem. Now, assume that the worst case had happened and after solving the reduced problem we got a unique optimal solution in which there exists no plane with some slack time. But, if there are empty flights covering counter loads, then we can still build up a good solution to the original problem, since the objective function value of the problem will be increased at most by $\bar{z}$, where,

$$\bar{z} = \left[\max_{i=1,..,p} \begin{pmatrix}\text{number of empty flights on} \\ \text{the route of plane } i\end{pmatrix}\right] \times [TU + TL] \quad (6.5)$$

Under such circumstances we can make the following proposal and claim that the solution at hand is still optimal to the original problem.

PROPOSAL 6.1

If the empty flights of the reduced problem covers all the dropped counter loads of the original problem, then one can build up a solution for the original problem utilizing the minimum total cost alternative solution of the reduced problem, and this solution will be the optimal solution for the original problem.

In order to prove this claim, assume that we have solved the original problem and this time we dropped the same loads to get a solution for the reduced problem. But this solution cannot be better than the solution obtained purely for the reduced problem. Since if it is so, then we should have found it while optimizing reduced problem. Hence the only difference in objective function values of the two problems can be caused from loading and unloading times which is independent of the routes of the planes, and in any case the mission times of planes are governed by their routes.

EXAMPLE 6.4: In order to analyze the effect of loads between the same kinds of points better, this time we have assigned a load between airports 2 and 1 and kept all other properties of the problem the same as in Example 6.3.

This load exploded the number of paths in generation phase ($P_1 = 27708$ $LC_{max} = 6$) and we did not execute the further steps of the BRP algorithm because of this huge number. One of the major reasons for this explosion is that airports 2 and 1 are very close to each other with respect to the other flights.

The existence of loads which require respectively shorter flight time brings troubles when number of paths generated considered. Let loads a, b, and c require more or less the same flight time and load d require relatively less flight time. Also let d_{max} be the maximum mission time given to generate paths. Now consider the following case. Let time to carry loads a and b $< d_{max}$. Under such a situation it happens most of the time that time to carry loads a, b, and c $> d_{max}$, but on the other hand usually time to carry loads a, b, and d $\leq d_{max}$. So, just a single load generates lots of combinations to be considered.

Now consider just the opposite case. That is, assume that load d requires relatively longer flight time. This case helps in all phases of the solution procedure. Since load d appears only on a small number of paths, the number of paths that have to be generated decreases. Also the search procedure automatically selects Method A for blocking, due to the unbalanced distribution of load frequencies. Thus, the number of paths in the first block decreases while computational effort during search procedure decreases.

EXAMPLE 6.5: This time in order to illustrate the explosion caused by counter loads better we have introduced another load between airports 1 and 5 as load 16 in addition to loads in Example 6.3. This resulted in $P_1 = 21622$ and $LC_{max} = 5$. By the use of the Proposal 6.1, we can build up the solution for this problem. Thus we did not carry on optimization.

EXAMPLE 6.6: Similarly, this example is designed for the same purpose as Example 6.5 and based on Example 6.1. Following counter loads are added to that problem.

Load 13 from airport 3 to airport 5.

Load 14 from airport 1 to airport 5.

Load 15 from airport 1 to airport 5.

Load 16 from airport 3 to airport 6.

This data resulted in $P_1 = 12769$ and $LC_{max} = 5$. But this time we cannot apply Proposal 6.1 since none of the empty flights of Example 6.1 covers load 16. So, we have to solve the problem without dropping load 16 once again.

EXAMPLE 6.7: This example is designed to analyze the effect of a central initial airport. This time all planes are assumed to be located at airport 4 initially and the other airports are assumed to be bases with no planes initially. This initial configuration of planes applied to 12 loads in Example 6.1. But maximum mission time to generate paths is increased to 495 minutes in this case. The results obtained are as follows;

$P_1 = 1034$ $P_2 = 264$ $LC_{max} = 3$

The alternative optimal solutions are as follows:

SOLUTION 1:

Plane	Initial airport	Loads carried	Time (in minutes)
1	4	7 - 12 - 3	470
2	4	5 - 1 - 2	470
3	4	4 - 8 - 10	460
4	4	9 - 6 - 11	455
		TOTAL:	1855

SOLUTION 2:

Plane	Initial airport	Loads carried	Time (in minutes)
1	4	7 - 3 - 6	460
2	4	5 - 1 - 2	470
3	4	9 -12 -11	460
4	4	4 - 8 -10	460
		TOTAL:	1850

SOLUTION 3:

Plane	Initial airports	Loads carried	Time (in minutes)
1	4	4 - 8 - 10	460
2	4	5 - 1 - 2	470
3	4	9 - 3 - 11	465
4	4	7 - 6 - 12	470
		TOTAL:	1865

$z^* = 470$ minutes and $P_3 = 198$.

Although there are loads starting from central airport 4, most of the loads start from airports 5 and 6, so planes at airport 4 should make an empty flight at first leg of their mission which naturally increases the optimal value of the problem.

The other example problems behaved similarly under this different initial configuration, and the increase in their objective function values are parallel to the increase in this case. In real life, there are other important factors which effect the choice of initial configuration in military applications. Although these factors are areas of interest in Operations Research, they are not related to our topic, so we have omitted these factors.

On the other hand, I want to emphasize on another factor which governs the efficiency of the solution procedure developed in this thesis, when the times required to carry loads are close each other.

Let there be M loads and P planes. Since times to carry each load do not vary much, most of the time the optimal solution of such problems results in minimum configuration. That is, $\overline{M}$ loads are assigned to each plane (where $\overline{M} = Q + 1$ and Q is defined by Definition 3.1). Since flight times are close each other, the times required to carry each set of $\overline{M}$ loads are close to each other. Hence, nearly all paths with load cardinality less than or equal to $\overline{M}$ have to be enumerated to catch an optimal solution. Thus, the minimum number of paths that have to be considered in the path list is $\overline{P}$ where,

$$\overline{P} = [(\sum_{r=1}^{\overline{M}} {}_{M}C_{r}) \times RP] \times \alpha , \qquad 0 < \alpha \leq 1 \tag{6.6}$$

$\overline{P}$ is the minimum number since there may be paths with load cardinality > $\overline{M}$ and time requirement $\leq d_{max}$. α is the ratio of paths with load cardinality $\leq \overline{M}$ and whose time requirement > d_{max}. As the times required to carry the loads approach each other, α tends to 1.

But the converse is not true, i.e. increasing range of mission times does not imply that $\alpha \to 0$. We are only trying to estimate the number of paths that have to be considered during optimization under different problem characteristics and we can simply assume $\alpha = 0.5$.

In all cases the problem gets much harder as $\overline{M}$ enlarges as compared with enlargements in M. We can follow this in Table 6.2 in detail.

TABLE 6.2 - Increase In Problem Complexity

M	P	RP	$\overline{M}$	$\overline{P}$
10	4	1	3	175α
20	4	1	5	21699α
20	6	1	4	6195α
15	5	1	3	575α
30	10	1	3	4525α
30	5	1	6	768211α
60	20	1	3	36050α
60	10	1	6	56049057α

6.2 GENERAL RESULTS

A computerized optimization for military cargo airplane routing is studied in this thesis. First the problem is formulated as a (0-1) linear program. Then two suggestions are given for two types of problems. Namely, the Bottleneck Routing Problem and the

Minimum Total Time Routing Problem. For the first problem a complete exact solution procedure is defined. But for the second one only a transformation is given. Both of these approaches aided to find out the routes of planes without taking into account the capabilities of service facilities at airports. In order to handle such constraints a heuristic procedure is suggested.

The algorithm developed for BRP is tested on various problems. These problems are relatively small problems as compared with real life problems. But the technique developed necessitates extensive computer usage. Although to obtain such facilities wide enough is a great problem in academic life, this is not the case in military applications. So in real life applications problems of reasonable size (See Table 1.1) can be solved.

There are several special cases of military airplane routing which are not explicitly discussed throughout the text. They are summarized as follows:

Most of the time the routing process is done upon essentially similar networks, and the boring part of the procedure defined is the generation, elimination and sorting of paths on these networks. In order to avoid this cumbursome business, the best thing to do is to estimate a representative network, which includes all airports and all possible loads between these airports. If generation, elimination and sorting processes are done on this network beforehand and resulting paths are stored somehow, then these paths can be utilized whenever necessary by eliminating the ones which are not desirable. This approach can be used while making sensitivity analysis over the solutions

That is, one can increase or decrease number of planes available or loads to be carried and can solve the problem just reinitiating the Bottleneck Routing Algorithm.

There can be cases where loads have to be transferred to points where no airports exist. In such cases, either the plane lands on an open area or simply drops the load. In any case the only difference is in the service times. One can assume any loading or unloading area as an airport, but utilize variable service times. The manipulation necessary to handle variable service times are explained throughout the text.

In real life applications, one important problem is, unit of shipment is not all the time a plane load. If that is the case, one must either develop a completely different approach or solve the problem by approximating all loads to unit plane loads, then analyze the solutions to get a solution to his original problem.

Commanders can state time windows. As an example, some loads should be carried between prespecified time intervals. The procedure in principle is not designed to handle such cases. But some modifications can be made at the search step of BRA in order to solve such problems.

The point that we have reached in this thesis is encouraging for further research. One can approach the problem in a totally, different way. That is, one could develop a formulation which does not necessitate the generation of simple paths at all. Some tour building techniques can be utilized instead of the set-partitioning approach. Also improvements on the algorithms developed within this study are possible. The stopping rules can be enlarged, so that one can be more sure about the feasible region.

The most valuable extension of this research would be designing a powerful technique which can handle splitable loads. Also one could attack solving military BRP in one step rather than first solving the routing problem and then satisfying the airport capacity constraints.

REFERENCES

1. Balas, E. and Christofides, N. (1981). "A Restricted Lagrangean Approach to the Travelling Salesman Problem", Mathematical Programming, 21, pp. 19-46.

2. Balas, E. and Padberg, M.W. (1972). "On the Set Covering Problem II: An Algorithm", Management Sciences Research Report No. 295, Carneige-Mellon University.

3. Balas, E. and Padberg, M.W. (1972). "On the Set Covering Problem", Ops. Res. 20, p. 1152.

4. Barr, R., Glover, F. and Klingman, D. (1974). "An Improved Version of the Out-of-Killer Method and a Comparative Study of Computer Codes", Mathematical Programming, 7(1), pp. 60-87.

5. Bellmore, M. and Hong, S. (1974). "Transformation of Multisalesmen Problem to the Standard Travelling Salesman Problem", Journal of the Association for Computing Machinery, 21, No. 3, pp. 500-504.

6. Berenguer, Y. (1979). "A Characterization of Linear Admissible Transformations for the m-Travelling Salesmen Problem", European Journal of Operational Research, 3, pp. 232-238.

7. Bodin, L. (1975). "A Taxonomic Structure for Vehicle Routing and Scheduling Problems", Computer Urban Soc., 1, pp. 11-29.

8. Bodin, L. and Berman, Lon. (1979). "Routing and Scheduling of School Buses by Computer", Transpn. Sci., 13, No. 2, pp. 113-129, New York.

9. Bodin, L. and Golden, B. (1981). "Classification in Vehicle Routing and Scheduling", Networks, 11, No. 2, pp. 97-108, New York.

10. Bodin, L., Golden, B., Assad, A. and Ball, M. (1981). "The State of the Art in the Routing and Scheduling of Vehicles and Crews", Working Paper, MS/S, No. 81-035, Virginia.

11. Bradley, G., Brown, G. and Graves, G. (1977). "Design and Implementation of Large Scale Primal Transshipment Algorithms", Management Sci., 24(1), pp. 1-34.

12. Chandy, K. and Lo, T. (1973). "The Capacitated Minimum Spanning Tree", Networks, 3(2), pp. 173-182.

13. Cheung, T.Y. (1980). "Computational Comparison of Eight Methods for the Maximum Network Flow Problem", ACM Transactions on Mathematical Software, 6(1), pp. 1-16.

14. Christofides, N. (1975), Graph Theory "An Algorithm Approach", London.

15. Christofides, N. (February 1976). "Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem", Report 388, Graduate School of Industrial Administration, Carneige Mellon University.

16. Christofides, N., Mingozzi, A. and Toth, P. (1981). "Exact Algorithms for the Vehicle Routing Problem, Based on Spanning Tree and Shortest Path Relaxations", Mathematical Programming, 20, pp. 255-282.

17. Clarke, G. and Wright, W. (1964). "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points", Ops. Res., 12, pp. 568-581.

18. Conway, R.W., Maxwell, W.C. and Miller, L.W. (1967). "Theory of Schedulling", Addison-Wesley, Reading, Mass.

19. Crowder, H. and Padberg, M. (May 1980). "Solving Large-Scale Symmetric Travelling Salesman Problems to Optimality", Management Sci., 26(5), pp. 495-509.

20. Cunningham, W. and Marsh, A. (July 1978). "A Primal Algorithm for Optimum Matching", Mathematical Programming Study, No. 8: Polyhedral Combinatorics, pp. 50-72.

21. Danielson, G. (1968). "On Finding the Simple Paths and Circuits in a Graph", IEEE Trans. on Circuit Theory, pp. 294-295.

22. Denardo, E. and Fox, b. (1979). "Shortest-Route Methods: 1. Reaching, Pruning and Bruckets", Ops. Res., 27(1), pp. 161-186.

23. Derigs, U. (April 1979). "A Shortest Augmenting Path Method for Solving Minimal Perfect Matching Problems", Technical Report, University of Cologne, Cologne, West Germany.

24. Derigs,U. and Kazakidis, G. (May 1979). "On Two Methods for Solving Minimal Perfect Matching Problems", Technical Report, University of Cologne, Cologne, West Germany.

25. Dial, R., Glover, F., Karney, D. and Klingman, D. (1979). "A Computational Analysis of Alternative Algorithms and Labeling Techniques for Finding Shortest Path Trees", Networks, 9(3), pp. 215-248.

26. Discenza, J.H. (1981). "A More Compact Formulation of the Symmetric Multiple Travelling Salesman Problem with Fixed Charges", Networks, 11, pp. 73-75.

27. Eilon, S., Watson-Gandy, C.D.T. and Christofides, N. (1971). Distribution Management, London.

28. Fratta, L. and Montanari, U. (1975). "A Vertex Elimination Algorithm for Enumerating all Simple Paths in a Graph", Networks, 5, pp. 151-177.

29. Gilsinn, J. and Witzgall, C. (1973). "A Performance Comparison of Labeling Algorithms for Calculating Shortest Path Trees", NBS Technical Note 777, National Bureau of Standards, Washington D.C.

30. Glover, F., Karney, D. and Klingman, D. (1974). "Implementation and Computational Comparisons of Primal, Dual and Primal-Dual Computer Codes for Minimum Cost Network Flow Problems", Networks, 4(3), pp. 191-212.

31. Golden, B. (June 1976). "Large Scale Vehicle Routing and Related Combinatorial Problems", Ph.D. Thesis, Operations Research Center, MIT.

32. Golden, B. and Ball, M. (1978). "Shortest Paths with Euclidean Distances: An Explanatory Model", Networks, 8(4), pp. 297-314.

33. Golden, B., Ball, M. and Bodin, L. (1981), "Current and Future Research Directions in Network Optimization", Comp. and Ops. Res. 8, pp. 71-81.

34. Golden, B. and Bodin, L. (1978). "Solving Large Scale Distribution-Routing Problems Efficiently", Proc. of 1978 Transportation and Logistics Educators Conference (R. House, ed.), pp. 11-14, Chicago.

35. Golden, B., Bodin, L., Doyle, T. and Stewart, W. (May-June 1980). "Approximate Travelling Salesman Algorithms", Ops. Res. 28(3), pp. 694-711.

36. Golden, B., Magnanti, T.L. and Nguyen, H.Q. (1977). "Implementing Vehicle Routing Algorithms", Networks, 7, pp. 113-148, Massachusett

37. Golden, B. and Richard, T. (1981). "Capacitated Arc Routing Problems", Networks, 11, pp. 305-315, New York.

38. Held, M. and Karp, R. (1970). "The Travelling Salesman Problem and Minimum Spanning Trees", Ops. Res., 18, pp. 1138-1162.

39. Held, M. and Karp, R. (1971). "The Travelling Salesman Problem and Minimum Spanning Tress, Part II", Mathematical Programming, 1, pp. 6-25.

40. Hong, S. and Padberg, M.W. (1977). "A Note on the Symmetric Multiple Travelling Salesman Problem with Fixed Charges", Ops. Res., 25, No. 5, pp. 871-874, New York.

41. Jensen, P.A. (1971). "Optimal Network Partitioning", Ops. Res., 19, p. 916.

42. Johnson, L.A. and Montgomery, D.C. (1974). "Operations Research in Production Planning, Scheduling and Inventory Control", John Wiley and Sons., Inc.

43. Kelton, W. and Law, A. (1978). "A Mean-Time Comparison of Algorithms for the All-Pairs Shortest-Path Problem with Arbitrary Arc Lengths", Networks, 8, pp. 97-106.

44. Kershenbaum, A. (1974). "Computing Capacitated Minimum Spanning Trees Efficiently", Networks, 4(4), pp. 299-310.

45. Kershenbaum, A. and Van Slyke, R. (1972). "Computing Minimum Spanning Trees Efficiently", Proceedings of ACM Annual Conference, pp. 518-527, Boston, Mass.

46. Kim, C. and Mac Donald, J. (1975). "A Minimal Spanning Tree and Approximate Tours for a Travelling Salesman", Comp. Sci. Technical Report, University of Maryland.

47. Knuth,D.E. (March 1975). "Sorting and Searching", The Art of Computer Programming, 3, Stanford University, Addison Wesley Publishing Company.

48. Kroft, D. (1967). "All Paths Through a Mazc", Proc. of IEEE, p.88.

49. Lin, S. (1965). "Computer Solutions of the Travelling Salesman Problem", Bell System Technical Journal, 44, pp. 2245-2269.

50. Lin, P.M. and Alderson, G.E. (1969). "Symbolic Network Functions by a Single Path-Finding Algorithm", Proc. of 7th Allerton Conference on Circuit and System Theory, p. 196.

51. Little, J., Murty, K., Sweeney, D. and Karel, C. (1963). "An Algorithm for the Travelling Salesman Problem", Ops. Res., 11(6), pp. 972-989.

52. Love, R.R. Jr. (1981). "Traffic Scheduling Via Benders Decomposition", Mathematical Programming Study, 15, pp. 102-124.

53. Michadu, P. (1972). "Exact Implicit Enumeration Method for Solving the Set-Partitioning Problem", IBM Jl. of Res. and Dev., 16, p. 573.

54. Miliotis, P. (1976). "Integer Programming Approaches to the Travelling Salesman Problem", Mathematical Programming, 10, pp. 367-378.

55. Miliotis, P. (1978). "Using Cutting Planes to Solve the Symmetric Travelling Salesman Problem", Mathematical Programming, 15, pp. 177-178.

56. Mulvey, J. (1978). "Testing of Large-Scale Network Optimization Program", Mathematical Programming, 15(3), pp. 291-314.

57. Norback, J. and Love, R. (1977). "Geometric Approaches to Solving the Travelling Salesman Problem", Management Sci., 23, pp. 1208-1223.

58. Padberg, M. and Hong, S. (1977). "On the Symmetric Travelling Salesman Problem: A Computational Study", T.J. Watson Research Report, IBM Research, Yorktown Heights.

59. Pape, U. (1974). "Implementation and Efficiency of Moore-Algorithms for the Shortest Route Problem", Mathematical Programming, 1, pp. 212-222.

60. Pierce, J.F. and Lasky, J.S. (1973). "Improved Combinatorial Programming Algorithms for a Class of all-zero-one Integer Programming Problems", Management Sci., 19, p. 528.

61. Psaraftis, H.N. (1980). "A Dynamic Programming Solution to the Single Vehicle Many-to-Many Immediate Request Dial-a-Ride Problem", Transpn. Sci., 14, No. 2, pp. 130-154, New York.

62. Rao, M.R. (1980), "A Note on the Multiple Travelling Salesmen Problem", Ops. Res., 28, No. 3, Part I, pp. 628-632, New York.

63. Richardson, R. (1976). "An Optimization Approach to Routing Aircraft", Transpn. Sci., 10, No. 1, pp. 52-71, New York.

64. Rosenkrantz, D., Stearns, R. and Lewis, P. (1977). "An Analysis of Several Heuristics for the Travelling Salesman Problem", SIAM J. Computing, 6, pp. 563-581.

65. Russel, R. (1977). "An Effective Heuristic for the m-Tour TSP with Some Side Conditions", ORSA, 25, pp. 517-524.

66. Salkin, H.M. and Koncal, R. (1971). "A Dual Algorithm for the Set Covering Problem", Dept. of O.R. Technical Memo., No. 250, Case Western University.

67. Shier, D. (1976). "Iterative Methods for Determining the k Shortest Paths in a Network", Networks, 6(3), pp. 205-229.

68. Shier, D. (1979). "On Algorithms for Finding the k Shortest Paths in a Network", Networks, 9(3), pp. 195-214.

69. Soumis, F., Ferland, J.A. and Rousseau, J.M. (1980). "A Model for Large-Scale Aircraft Routing and Scheduling Problems", Transpn. Res.-B, 14B, pp. 191-201, Great Britain.

70. Stein, D.M. (1978). "Scheduling Dial-a-Ride Transportation Systems", Transpn. Sci., 12, No. 3, pp. 232-249, New York.

71. Ulusoy, G. (1981). "Taşıt Güzergahı Belirleme Problemleri ve Bazı Yeni Algoritmalar", Doçentlik Tezi, Boğaziçi Üniversitesi Mühendislik Fakültesi.

72. Ulusoy, G. (1983). "Routing in Strategic Airlift: A Study in Bottleneck Routing", Dept. of Ind. Eng'g, Boğaziçi University.

73. Webb, M. (1971). "Some Methods of Producing Approximate Solutions to Travelling of Cities", Ops. Res. Q., 22(1), pp. 49-66.

APPENDICES

APPENDIX A

The formulations discussed in various chapters of this thesis are applied to some test problems, and these problems are solved with the techniques suggested.

We can summarize these formulations as follows:

i. <u>FORMULATION I</u>:

Set-theoretic formulation of BRP, solved by the algorithm suggested in Chapter 3.

ii. <u>FORMULATION II</u>:

Set-theoretic formulation of BRP with minimum total cost requirement solved by the algorithm suggested in Chapter 3.

iii. <u>FORMULATION III</u>:

MDMTSP formulation of MCRP by using exactly the given number of planes, solved by Little's Branch and Bound algorithm.

iv. <u>FORMULATION IV</u>:

MDMTSP formulation of MCRP by using less than or equal to the given number of planes, solved by Little's Branch and Bound algorithm.

v. FORMULATION V:

MDMTSPF formulation of MCRP with fixed costs of planes, solved by Little's Branch and Bound algorithm.

Following format is used to express results:

Plane number: Initial airport: Loads carried: Time

NOTES: i) "+" sign besides total times indicates that, that results correspond to the one which is obtained at a reasonable time on UNIVAC 1106. Further improvement possible if a better TSP algorithm occupied.

ii) The unit of time is taken as minutes.

EXAMPLE A

$$D = \begin{array}{c|cccc|} 1 & - & 30 & 65 & 40 \\ 2 & 30 & - & 30 & 35 \\ 3 & 65 & 30 & - & 20 \\ 4 & 40 & 35 & 20 & - \end{array}$$

Loading time : 10

Unloading time: 5

LOADS:

Number	Starting airport	Ending airport
1	1	2
2	4	1
3	2	4
4	4	3
5	4	2
6	3	1
7	3	4

INITIAL LOCATIONS OF PLANES:

Airport	Number of planes
1	0
2	0
3	1
4	1

Minimum Total Flight:		245
Total Load Unload	:	105
Minimum Total Time	:	350

FORMULATION I

Plane 1 : 4: 4-6-1	:	160
Plane 2 : 3: 7-5-3-2	:	190
TOTAL		350

FORMULATION II

Plane 1 : 4: 4-6-1	:	160
Plane 2 : 3: 7-5-3-2	:	190
TOTAL		350

FORMULATION III

Plane 1 : 3: 7-4-6-1	:	195
Plane 2 : 3: 5-3-2	:	155
TOTAL		350

FORMULATION IV

Plane 1 : 3: 6-1	:	125
Plane 2 : 4: 4-7-5-3-2	:	225
TOTAL		350

FORMULATION V

a) Given:

Airport	Fixed Cost of a plane
3	100
4	100

Plane 1: 4: 4-7-5-6-1-3-2 : 380

TOTAL 380

Thus, Total cost: 480

b) Given:

Airport	Fixed Cost of a plane
3	200
4	100

Plane 1: 4: 2-1-3-4-7-5-6 : 380

TOTAL 380

Thus, Total cost: 480

c) Given:

Airport	Fixed Cost of a plane
3	100
4	200

Plane 1: 3: 6-1-3-4-7-5-2 : 385

TOTAL 385

Thus, Total cost: 485

DISCUSSION

Although formulations III and IV have the same objective function values, the resulting optimal solutions assigned planes to different set of loads, that is due to the branching nature of Little's Branch and Bound algorithm on similar cost matrices. Also, cases a and b of formulation V are different because of the same reason. Indeed same phenomenon is observed in most of the examples but will not be discussed from now on.

Also note that, there is only one alternative optimal solution to BRP. So, formulation I and II yielded the same results.

EXAMPLE B

Same D matrix as in Example A. Load and unload times are also the same.

LOADS:

Number	Starting airport	Ending airport
1	1	2
2.	4	1
3	2	4
4	4	3
5	4	2
6	3	1
7	3	4
8	3	2
9	2	1
10	4	3

Same as Example A

INITIAL LOCATIONS OF PLANES:

Airport	Number of planes
1	0
2	0
3	0
4	3

Minimum Total Flight: 325

Total Load Unload : 150

Minimum Total Time : 475

FORMULATION I

Plane 1: 4: 7-10-6 : 170

Plane 2: 4: 5- 3-2 : 155

Plane 3: 4: 4- 8-9-1 : 170

TOTAL 495

FORMULATION II

Plane 1: 4: 7-10-6	: 170
Plane 2: 4: 5- 3-2	: 155
Plane 3: 4: 4- 8-9-1	: 170
TOTAL	495

FORMULATION III

Plane 1: 4: 2	: 55
Plane 2: 4: 4-8-9-1-6	: 280
Plane 3: 4: 5-3-7-10	: 190
TOTAL	525 +

FORMULATION IV

Plane 1: 4: 4-7-10-8-9-1	: 240
Plane 2: 4: 2	: 55
Plane 3: 4: 5-3-6	: 200
TOTAL	495

FORMULATION V

Given fixed cost of plane at port 4 is 100.

Plane 1: 4: 2-4-7-10-8-9-1-3-5-6: 545

Thus, Total Cost: 645+

DISCUSSION

Although the optimal solution to formulation IV is found with 3 planes, formulation III failed in achieving this result in a reasonable time. If run had not been terminated then, formulation III would eventually reach that solution also. In several examples such a case occurred, which indicates that Little's Branch and Bound algorithm functioning better on formulation IV.

EXAMPLE C

Same D matrix as in Example A and B. Load and unload times are also the same.

LOADS:

Number	Starting airport	Ending airport
1	1	2
2	4	1
3	2	4
4	4	3
5	4	2
6	3	1
7	3	4
8	3	2
9	2	1
10	4	3
11	1	3
12	1	4

Same as Example B

INITIAL LOCATIONS OF PLANES:

Airport	Number of planes
1	0
2	0
3	2
4	4

Minimum Total Flight: 430

Total Load Unload : 180

Minimum Total Time : 610

FORMULATION I

Plane 1: 4: 11	: 120
Plane 2: 4: 2-12	: 110
Plane 3: 4: 10- 6	: 115
Plane 4: 3: 9- 1	: 120
Plane 5: 4: 5- 3	: 100
Plane 6: 3: 7- 4-8	: 115
TOTAL	680

FORMULATION II

Plane 1: 4: 11	: 120
Plane 2: 4: 2-12	: 110
Plane 3: 4: 10- 6	: 115
Plane 4: 3: 9- 1	: 120
Plane 5: 4: 5- 3	: 100
Plane 6: 3: 7- 4-8	: 115
TOTAL	680

FORMULATION III

Plane 1: 3: 8-3-10	: 130
Plane 2: 4: 6-11	: 180
Plane 3: 4: 5	: 50
Plane 4: 4: 4-7	: 70
Plane 5: 4: 2-1	: 100
Plane 6: 3: 9-12	: 130
	660+

FORMULATION IV

Plane 1: 4: 4-6-11-8-9-12-10-7-5-3-2-1 : 610

Thus, Total Cost: 610

FORMULATION V

a) Given cost of a plane at airport 3 is 100.

Given cost of a plane at airport 4 is 100.

Plane 1: 3: 6-11-8-9-12-10-7-4-5-3-2-1: 630

Thus, Total Cost : 730 +

b) Given cost of a plane at airport 3 is 100.

Given cost of a plane at airport 4 is 200.

Plane 1: 3: 6-11-8-9-12-10-7-4-5-3-2-1: 630

Thus, Total Cost : 730 +

c) Given cost of a plane at airport 3 is 200.

Given cost of a plane at airport 4 is 100.

Plane 1: 4: 4-6-11-8-9-12-10-7-5-3-2-1: 610

Thus, Total Cost : 710

d) Given fixed cost of a plane at airport 3 is 5.

Given fixed cost of a plane at airport 4 is 10.

Plane 1: 3: 6-11-8-9-12-10-7-4-5-3-2-1: 630

Thus, Total Cost: 635 +

e) Given fixed cost of a plane at airport 3 is 2.

Given fixed cost of a plane at airport 4 is 2.

Plane 1: 3: 6-11-8-9-12-10-7-4-5-3-2-1: 630

Thus, Total Cost : 635 +

DISCUSSION

In this case, formulation IV resulted with an optimal solution of value 610 and using only one plane. But again formulation III failed in achieving the optimal solution in a reasonable time. The optimal solution to formulation III has to have a value greater than or equal to 610 as discussed in Chapter 5. In this case we can built up the optimal solution to formulation III by simply splitting the route obtained as a result of formulation IV as follows:

Plane 1:	4:	4	:	35
Plane 2:	3:	6-11	:	160
Plane 3:	3:	8-9-12	:	145
Plane 4:	4:	10-7	:	70
Plane 5:	4:	5-3	:	100
Plane 6:	4:	2-1	:	100
			TOTAL	610

By this way the optimal solution to formulation III has been obtained with optimal value 610. In the following example similar decomposition are applicable.

EXAMPLE D

Same D matrix as Examples A, B, and C. Load and unload times are the same.

LOADS:

Number	Starting airport	Ending airport
1	1	2
2	4	1
3	2	4
4	4	3
5	4	2
6	3	1
7	3	4
8	3	2
9	2	1
10	4	3
11	1	3
12	1	4
13	3	2
14	3	2
15	3	4

Same as Example C

INITIAL LOCATIONS OF PLANES:

Airport	Number of planes
1	0
2	0
3	3
4	4

Minimum Total Flight: 510

Total Load Unload : 225

Minimum Total Time : 735

FORMULATION I

Plane 1: 4: 2-11	: 135
Plane 2: 3: 10-6	: 135
Plane 3: 4: 5-12	: 135
Plane 4: 3: 13-9-1	: 135
Plane 5: 3: 8-3-4	: 130
Plane 6: 4: 7-14	: 120
Plane 7: 4: 15	: 55
TOTAL	845

FORMULATION II

Plane 1: 4: 2-11	: 135
Plane 2: 3: 6-12	: 135
Plane 3: 3: 13-9-1	: 135
Plane 4: 3: 15-5-3	: 135
Plane 5: 4: 7-10-8	: 135
Plane 6: 4: 4-14	: 80
TOTAL	755

FORMUTAION III

Plane 1: 3: 7	: 35
Plane 2: 3: 6-1-11	: 235
Plane 3: 4: 13	: 85
Plane 4: 4: 2-5-3-10	: 220
Plane 5: 3: 8-9-12	: 145
Plane 6: 4: 14	65
Plane 7: 4: 4-15	: 70
TOTAL	845+

FORMULATION IV

Plane 1: 3: 6-11-15-10-14-9-12-5: 425

Plane 2: 4: 2-1 : 100

Plane 3: 3: 7-4-13-3 : 165

Plane 4: 3: 8 : 45

TOTAL 735

FORMULATION V

a) Fixed cost of a plane at airport 3 is 100.
Fixed cost of a plane at airport 4 is 100.

Plane 1: 3: 6-11-15-10-14-9-12-5-13-3-2-1-7-4-8: 795

Thus, Total Cost : 895+

b) Fixed cost of a plane at airport 3 is 200.
Fixed cost of a plane at airport 4 is 100.

Plane 1: 4: 6-11-15-10-14-9-12-5-13-3-2-1-7-4-8: 815

Thus, Total Cost : 915+

DISCUSSION

Note that the optimal solution to formulation II utilizes only 6 planes, although we have 7 available. This case is mentioned in Chapter 3. That is, we can reach to the optimal solution to BRP by using less than the given number of planes while applying the algorithm developed in this thesis.

EXAMPLE D WITH THE PLANE AVAILABILITY AS FOLLOWS:

Airport	Number of planes
1	0
2	0
3	2
4	4

FORMULATION I

Plane 1: 4: 2-11	: 135
Plane 2: 3: 6-12	: 135
Plane 3: 3: 13-9-1	: 135
Plane 4: 4: 5-3-10	: 135
Plane 5: 4: 4-15-8	: 135
Plane 6: 4: 7-14	: 120
TOTAL	795

FORMULATION III

Plane 1: 3: 14-9-12	: 145
Plane 2: 4: 6-11	: 180
Plane 3: 4: 4-13-3-7-10	: 220
Plane 4: 4: 2-1-8	: 175
Plane 5: 4: 5	: 50
Plane 6: 3: 15	: 35
TOTAL	805+

FORMULATION IV

Plane 1: 3: 6-11-15-10-14-9-12-5	:	425
Plane 2: 4: 2-1	:	100
Plane 3: 3: 7-4-13-3-8	:	230
TOTAL		755+

FORMULATION V

a) Fixed cost of a plane at airport 3 is 100.

Fixed cost of a plane at airport 4 is 100.

Plane 1: 3: 6-11-15-12-14-9-12-5-13-3-2-1-7-4-8: 795

Thus, Total Cost : 895+

b) Fixed cost of a plane at airport 3 is 200.

Fixed cost of a plane at airport 4 is 100.

Plane 1: 4: 6-11-15-10-14-9-12-5-13-3-2-1-7-4-8: 815

Thus, Total Cost : 915

EXAMPLE D WITH THE PLANE AVAILABILITY AS FOLLOWS:

Airport	Number of planes
1	0
2	0
3	1
4	5

FORMULATION I

Plane 1: 4: 2-11	: 135
Plane 2: 3: 6-12	: 135
Plane 3: 4: 5-9-1	: 135
Plane 4: 4: 10-8-3	: 130
Plane 5: 4: 13-14	: 140
Plane 6: 4: 4-7-15	: 125
TOTAL	800

EXAMPLE D WITH THE PLANE AVAILABILITY AS FOLLOWS:

Airport	Number of planes
1	0
2	0
3	3
4	3

FORMULATION I

Plane 1: 4: 2-11	: 135
Plane 2: 3: 6-12	: 135
Plane 3: 3: 13-9-1	: 135
Plane 4: 4: 5-3-10	: 135
Plane 5: 4: 4-15-8	: 135
Plane 6: 3: 7-14	: 100
TOTAL	775

EXAMPLE E

$$D = \begin{array}{c|ccccc|} 1 & - & 30 & 65 & 40 & 90 \\ 2 & 30 & - & 30 & 35 & 60 \\ 3 & 65 & 30 & - & 20 & 20 \\ 4 & 40 & 35 & 20 & - & 35 \\ 5 & 30 & 60 & 20 & 35 & - \end{array}$$

Loading time : 10

Unloading time: 5

Actually airport 5 is added to Example A.

LOADS:

Number	Starting airport	Ending airport
1	1	2
2	4	1
3	2	4
4	4	3
5	4	2
6	3	1
7	3	4
8	3	2
9	2	1
10	4	3
11	1	3
12	1	4
13	3	2
14	3	2
15	3	4
16	5	1
17	5	3

Same as Example C

INITIAL LOCATIONS OF PLANES:

Airport	Number of planes
1	0
2	0
3	0
4	5
5	2

Minimum Total Flight: 560

Total Load Unload : 255

Minimum Total Time : 815

<u>FORMULATION I</u>

Plane 1: 4: 2-11	: 135
Plane 2: 4: 10-6	: 115
Plane 3: 5: 16-1-9	: 135
Plane 4: 4: 5-12	: 135
Plane 5: 4: 4-13-3	: 130
Plane 6: 5: 17-7-8	: 135
Plane 7: 4: 15-14	: 120
TOTAL	905

<u>FORMULATION II</u>

Plane 1: 4: 2-11	: 135
Plane 2: 4: 10-6	: 115
Plane 3: 5: 16-1-9	: 135
Plane 4: 4: 5-12	: 135
Plane 5: 4: 4-13-4	: 130
Plane 6: 5: 17-7-8	: 135
Plane 7: 4: 15-14	: 120
TOTAL	905

FORMULATION III

Plane 1: 4: 4-13-3-10	: 165
Plane 2: 4: 6-1-11	: 255
Plane 3: 4: 7	: 55
Plane 4: 4: 2-5-14-9-12	: 320
Plane 5: 4: 8-15	: 130
Plane 6: 5: 16	: 45
Plane 7: 5: 17	: 35
TOTAL	1005+

FORMULATION IV

Plane 1: 4: 2	: 55
Plane 2: 5: 16-12-5	: 150
Plane 3: 5: 17-15-10-13-3-7-6-11-14-9-1	: 570
Plane 4: 4: 4-8	: 80
TOTAL	855+

FORMULATION V

a) Given:

Airport	Fixed cost of a plane
4	100
5	100

Plane 1: 5: 17-16-12-5-13-3-6-11-15-12-14-9-1-7-2-4-8: 955

Thus, Total Cost :1055+

b) Given:

Airport	Fixed cost of a plane
4	200
5	100

Plane 1: 5: 17-16-12-5-13-3-6-11-15-10-14-9-1-7-2-4-8: 955

Thus, Total Cost : 1055+

c) Given:

Airport	Fixed cost of a plane
4	100
5	200

Plane 1: 4: 6-1-7-2-4-15-10-14-9-11-17-16-12-5-13-3-8: 995

Thus, Total Cost : 1095+

d) Given:

Airport	Fixed cost of a plane
4	10
5	5

Plane 1: 5: 16-12-5-6-11-14-9-1 : 475

Plane 2: 5: 17-15-10-13-3-7-4-8-2 : 425

TOTAL 900

Thus, Total Cost: 910+

EXAMPLE F

$$D = \begin{array}{c|cccccc|} 1 & - & 30 & 65 & 40 & 90 & 85 \\ 2 & 30 & - & 30 & 35 & 60 & 95 \\ 3 & 65 & 30 & - & 20 & 20 & 100 \\ 4 & 40 & 35 & 20 & - & 35 & 75 \\ 5 & 90 & 60 & 20 & 35 & - & 110 \\ 6 & 85 & 95 & 100 & 65 & 110 & - \end{array}$$

Loading time : 10

Unloading time: 5

LOADS:

Number	Starting airport	Ending airport
1	1	2
2	4	1
3	2	4
4	4	3
5	4	2
6	3	1
7	3	4
8	3	3
9	2	1
10	4	3
11	1	3
12	1	4
13	3	2
14	3	2
15	3	4
16	5	1
17	5	3
18	6	1

INITIAL LOCATIONS OF PLANES:

Airport	Number of planes
1	0
2	0
3	3
4	2
5	2
6	0

Minimum Total Flight: 705

Total Load Unload : 270

Minimum Total Time : 975

FORMULATION I

Plane 1: 4: 18	: 175
Plane 2: 3: 15-16	: 175
Plane 3: 5: 14-11	: 175
Plane 4: 3: 6-2	: 175
Plane 5: 4: 5-12-4	: 170
Plane 6: 5: 17-8-9-1	: 170
Plane 7: 3: 7-10-13-3	: 165
TOTAL	1205

FORMULATION II

Plane 1: 4: 18	: 175
Plane 2: 3: 15-16	: 175
Plane 3: 5: 14-11	: 175
Plane 4: 3: 6-2	: 175
Plane 5: 4: 5-12-4	: 170
Plane 6: 5: 17-8-9-1	: 170
Plane 7: 3: 7-10-13-3	: 165
TOTAL	1205

FORMULATION III

Plane 1: 3: 6-2-4-14	: 295
Plane 2: 5: 16-11	: 185
Plane 3: 3: 8-9-1-3-13	: 250
Plane 4: 4: 18-12	: 230
Plane 5: 4: 10	: 35
Plane 6: 3: 7-5-15	: 150
Plane 7: 5: 17	: 35
TOTAL	1180+

FORMULATION IV

Plane 1: 3: 7-5	: 85
Plane 2: 5: 16-11-14-9-1	: 320
Plane 3: 4: 2	: 55
Plane 4: 3: 6	: 80
Plane 5: 5: 17-15-18-12-10-13-3-4-8	: 510
TOTAL	1050

FORMULATION V

a) Given:

Airport	Fixed cost of a plane
3	100
4	100
5	100

Plane: 1: 5: 17-16-11-15-18-12-10-14-9-1-7-5-13-3-6-2-4-8: 1230

Thus, Total Cost : 1330+

b) Given:

Airport	Fixed cost of a plane
3	100
4	200
5	100

Plane 1: 5: 17-16-11-15-18-12-10-14-9-1-7-5-13-3-6-2-4-8: 1230

Thus, Total Cost : 1330+

c) Given:

Airport	Fixed cost of a plane
3	100
4	100
5	200

Plane 1: 3: 7-4-13-3-17-16-11-15-18-12-10-14-9-1-6-2-4-8: 1245

Thus, Total Cost : 1345+

d) Given:

Airport	Fixed cost of a plane
3	0
4	5
5	10

Plane 1: 3: 2	: 75
Plane 2: 3: 7-5	: 85
Plane 3: 5: 16-11-14-9-1	: 320
Plane 4: 3: 6	: 80
Plane 5: 5: 17-15-18-12-10-13-3-4-8:	510
TOTAL	1070

Thus, Total Cost: 1090+

EXAMPLE G

$$D = \begin{array}{c|ccccc|} 1 & - & 30 & 40 & 50 & 65 \\ 2 & 30 & - & 55 & 35 & 25 \\ 3 & 40 & 55 & - & 30 & 70 \\ 4 & 50 & 35 & 30 & - & 20 \\ 5 & 65 & 25 & 70 & 20 & - \end{array}$$

Loading time : 45

Unloading time: 25

LOADS:

Number	Starting airport	Ending airport
1	1	2
2	4	2
3	2	3
4	3	1
5	1	4
6	3	4
7	5	4
8	2	5
9	1	5
10	5	3
11	4	3
12	4	3
13	3	2
14	4	2
15	4	2
16	2	4
17	2	1
18	1	3
19	4	5
20	4	1

INITIAL LOCATIONS OF PLANES:

Airport	Number of planes
1	4
2	0
3	0
4	4
5	0

Minimum Total Flight: 780

Total Load Unload : 1400

Minimum Total Time : 2980

FORMULATION I

Plane 1: 4: 14-17-1	: 305
Plane 2: 4: 15-16-19	: 300
Plane 3: 4: 11-6-12	: 300
Plane 4: 4: 2-8-7	: 290
Plane 5: 1: 3-20	: 305
Plane 6: 1: 4-9	: 285
Plane 7: 1: 5-10	: 280
Plane 8: 1: 18-13	: 235
TOTAL	2300

FORMULATION II

Plane 1: 4: 14-17-1	: 305
Plane 2: 4: 15-16-19	: 300
Plane 3: 4: 11-6-12	: 300
Plane 4: 4: 11-6-12	: 290
Plane 5: 1: 3-13	: 280
Plane 6: 1: 9-10	: 275
Plane 7: 1: 5-20	: 240
Plane 8: 1: 18-4	: 220
TOTAL	2210

FORMULATION III

Plane 1:	1:	5-16	: 225
Plane 2:	1:	18	: 110
Plane 3:	4:	12-13-17	: 325
Plane 4:	4:	11-6-19	: 290
Plane 5:	4:	14	: 105
Plane 6:	1:	9-10-4	: 385
Plane 7:	1:	1-3-16	: 385
Plane 8:	4:	2-8-7-20	: 410
		TOTAL	2235+

FORMULATION IV

Plane 1:	1:	1	: 100
Plane 2:	4:	2-3	: 230
Plane 3:	4:	11-4-5-14-16-20-18-13-8	: 990
Plane 4:	4:	12-6-15-17-9-7-19-10	: 860
		TOTAL	2180

FORMULATION V

a) Given:

Airport	Fixed cost of a plane
1	100
4	100

Plane 1: 1: 1-3-19-10-15-17-9-7-14-16-20-18-13-8-12-6-11-4-5-2: 2260

Thus, Total Cost : 2360+

b) Given:

Airport	Fixed cost of a plane
1	200
4	100

Plane 1: 4: 2-1-3-11-4-5-12-6-15-17-9-7-14-16-20-18-13-8-19-10: 2260

Thus, Total Cost : 2360+

APPENDIX B

	Combination Values of M and r					
M/r	1	2	3	4	5	6
10	10	45	120	210	252	210
11	11	55	165	330	462	462
12	12	66	220	495	792	924
13	13	78	286	715	1287	1716
14	14	91	364	1001	2002	3003
15	15	105	455	1365	3003	5005
16	16	120	560	1820	4368	8008
17	17	136	680	2380	6188	12376
18	18	153	816	3060	8568	18564
19	19	171	969	3876	11628	27132
20	20	190	1140	4845	15504	38760
21	21	210	1330	5985	20349	54264
22	22	231	1540	7315	26334	74613
23	23	253	1771	8855	33649	100947
24	24	276	2024	10626	42504	134596
25	25	300	2300	12650	53130	177100
26	26	325	2600	14950	65780	230230
27	27	351	2925	17550	80730	296010
28	28	378	3276	20475	98280	376740
29	29	406	3654	23751	118755	475020
30	30	435	4060	27405	142506	593775
31	31	465	4495	31465	169911	736281
32	32	496	4960	35960	201376	906192
33	33	528	5456	40920	237336	1107568

Permutation Values of M and r

M/r	1	2	3	4	5	6
10	10	90	720	5040	30240	151200
11	11	110	990	7920	55440	332640
12	12	132	1320	11880	95040	665280
13	13	156	1716	17160	154440	1235520
14	14	182	2184	24024	240240	2162160
15	15	210	2730	32760	360360	3603600
16	16	240	3360	43680	524160	5765760
17	17	272	4080	57120	742560	8910720
18	18	306	4896	73440	1028160	13366080
19	19	342	5814	93024	1395360	19535040
20	20	380	6840	116280	1860480	27907200
21	21	420	7980	143640	2441880	39070080
22	22	462	9240	175560	3160080	53721360
23	23	506	10626	212520	4037880	72681840
24	24	552	12144	255024	5100480	96909120
25	25	600	13800	303600	6375600	127512000

APPENDIX C

C.1 SIMPLE PATH GENERATION

C.1.1 Definitons

a) NN: number of nodes

b) S: the source node

c) T: the terminal node

d)

$$W(I,J) = \begin{cases} \text{length of arc } (I,J); & \text{if dones I and J are directly connected by an arc.} \\ \infty & \text{; otherwise.} \end{cases}$$

e)

$$MARK(I) = \begin{cases} 1; & \text{if node I is unmarked.} \\ 0; & \text{if node I is marked.} \end{cases}$$

f) KTH : Current path counter shows the number of simple paths yet generated.

g) DMAX: Maximum distance to generate paths.

h) LEN : Length of current partial path.

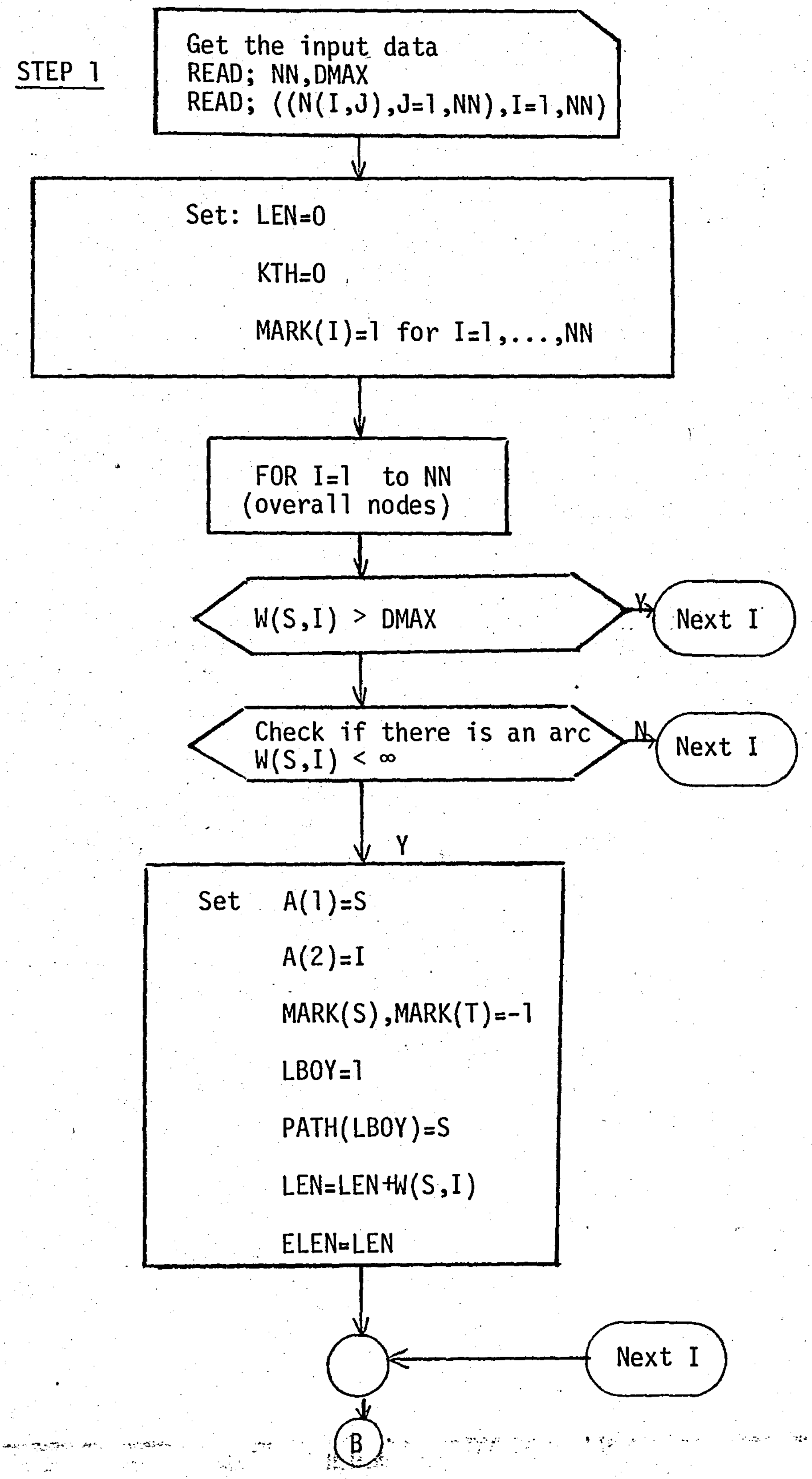
i) PATH: Array that contains current partial path.

j) LBOY: Number of elements on current partial path.

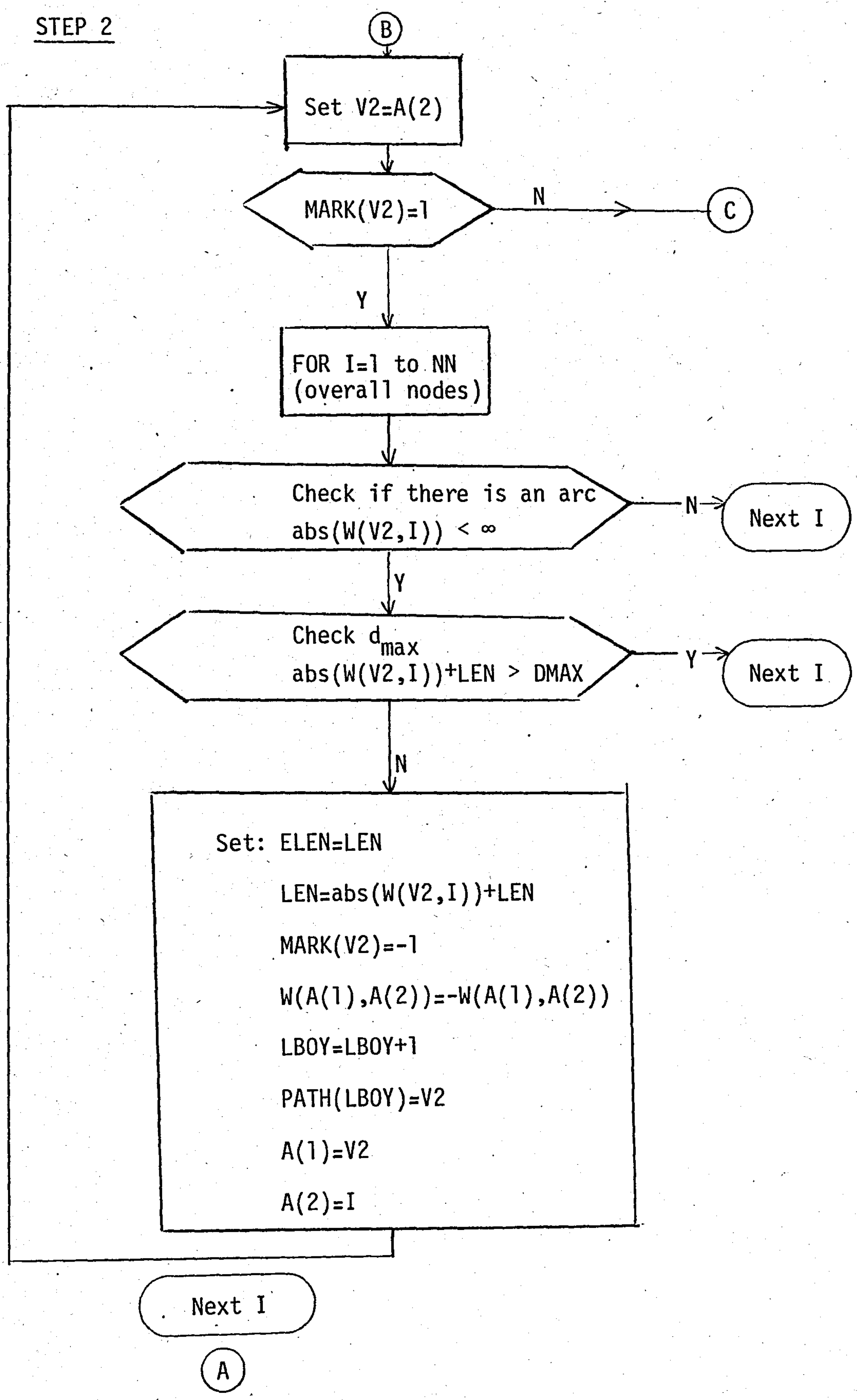
k) A(1): Starting node of current arc.

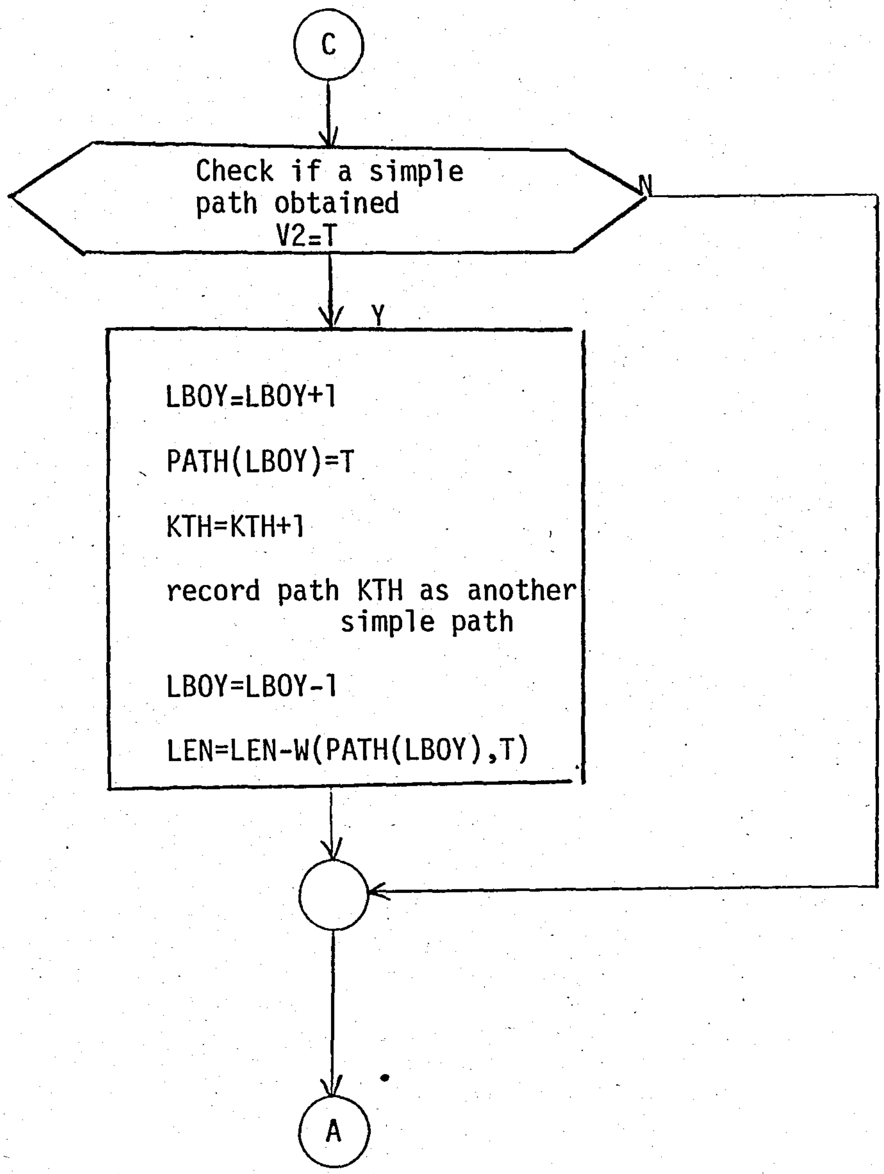
l) A(2): Ending node of current arc.

C.1.2 Flowchart 1

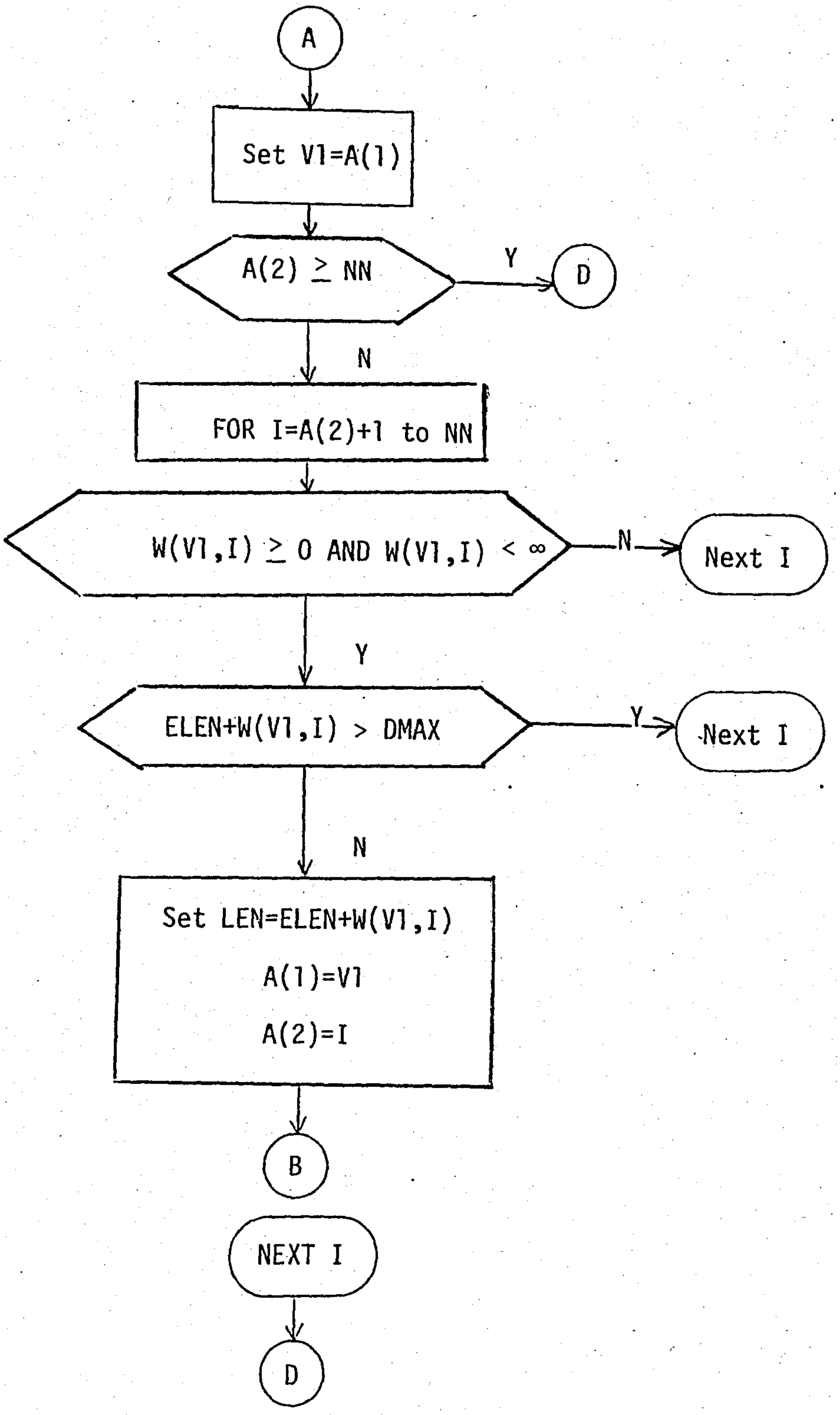


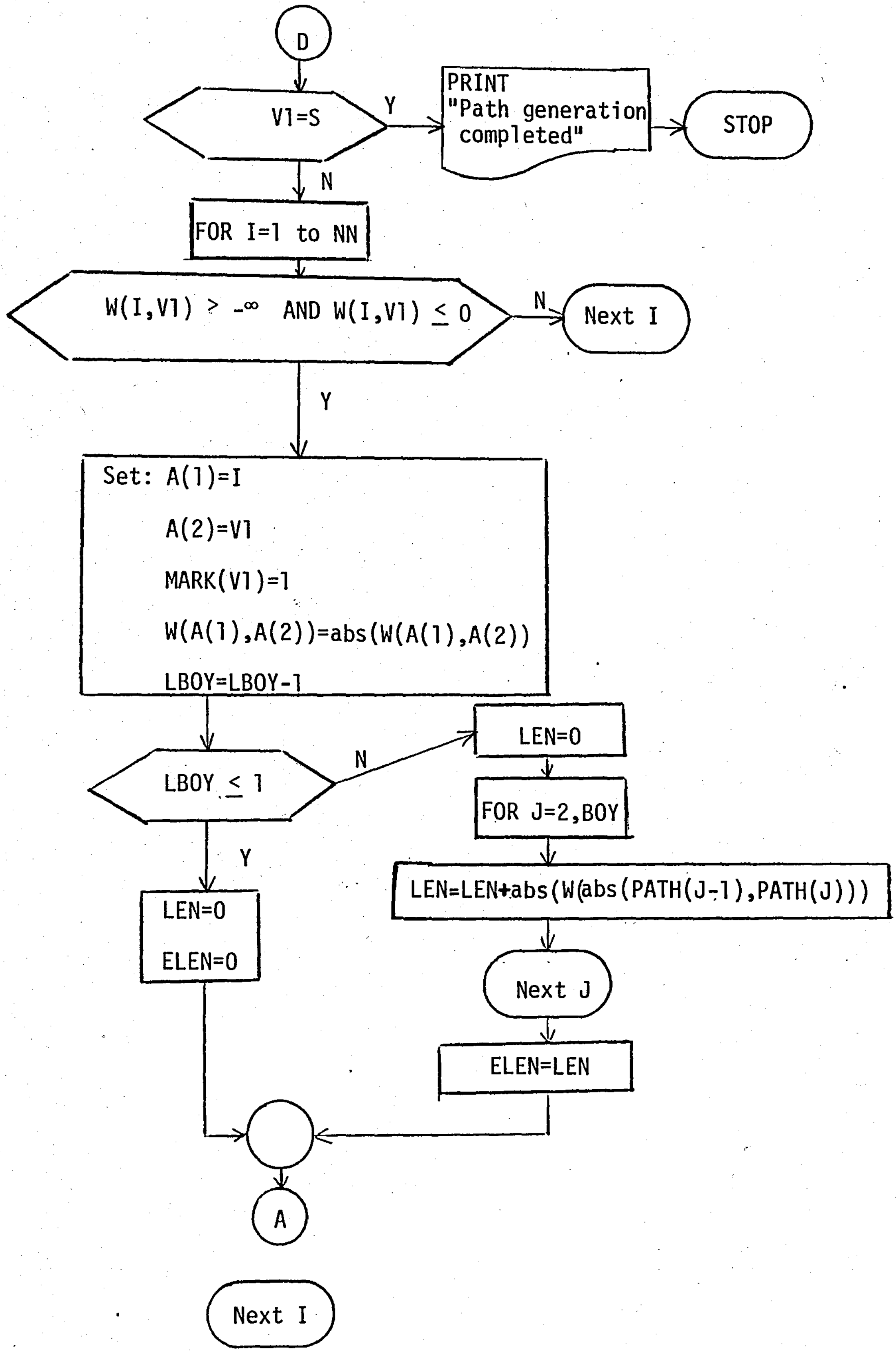
STEP 2


B
Set V2=A(2)
MARK(V2)=1
N
C
Y
FOR I=1 to NN (overall nodes)
Check if there is an arc abs(W(V2,I)) < ∞
N
Next I
Y
Check d_max abs(W(V2,I))+LEN > DMAX
Y
Next I
N
Set: ELEN=LEN
LEN=abs(W(V2,I))+LEN
MARK(V2)=-1
W(A(1),A(2))=-W(A(1),A(2))
LBOY=LBOY+1
PATH(LBOY)=V2
A(1)=V2
A(2)=I
Next I
A


C
Check if a simple path obtained V2=T
N
Y
LBOY=LBOY+1
PATH(LBOY)=T
KTH=KTH+1
record path KTH as another simple path
LBOY=LBOY-1
LEN=LEN-W(PATH(LBOY),T)
A

STEP 3


A
Set V1=A(1)
A(2) ≥ NN
Y
D
N
FOR I=A(2)+1 to NN
W(V1,I) ≥ 0 AND W(V1,I) < ∞
N
Next I
Y
ELEN+W(V1,I) > DMAX
Y
Next I
N
Set LEN=ELEN+W(V1,I)
A(1)=V1
A(2)=I
B
NEXT I
D

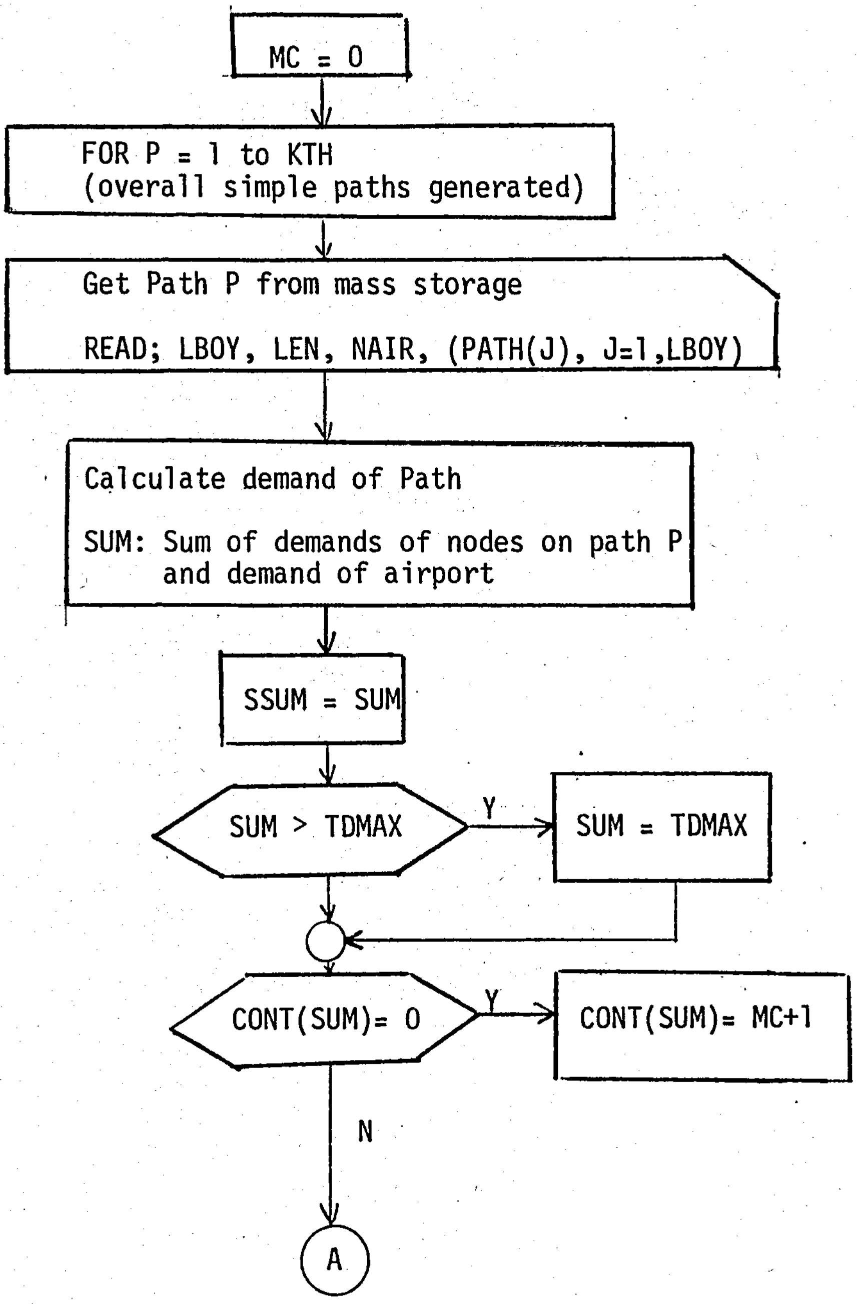

D
V1=S
Y
PRINT "Path generation completed"
STOP
N
FOR I=1 to NN
W(I,V1) > -∞ AND W(I,V1) ≤ 0
N
Next I
Y
Set: A(1)=I
A(2)=V1
MARK(V1)=1
W(A(1),A(2))=abs(W(A(1),A(2))
LBOY=LBOY-1
LBOY ≤ 1
N
LEN=0
FOR J=2,BOY
LEN=LEN+abs(W(abs(PATH(J-1),PATH(J)))
Next J
ELEN=LEN
Y
LEN=0
ELEN=0
A
Next I

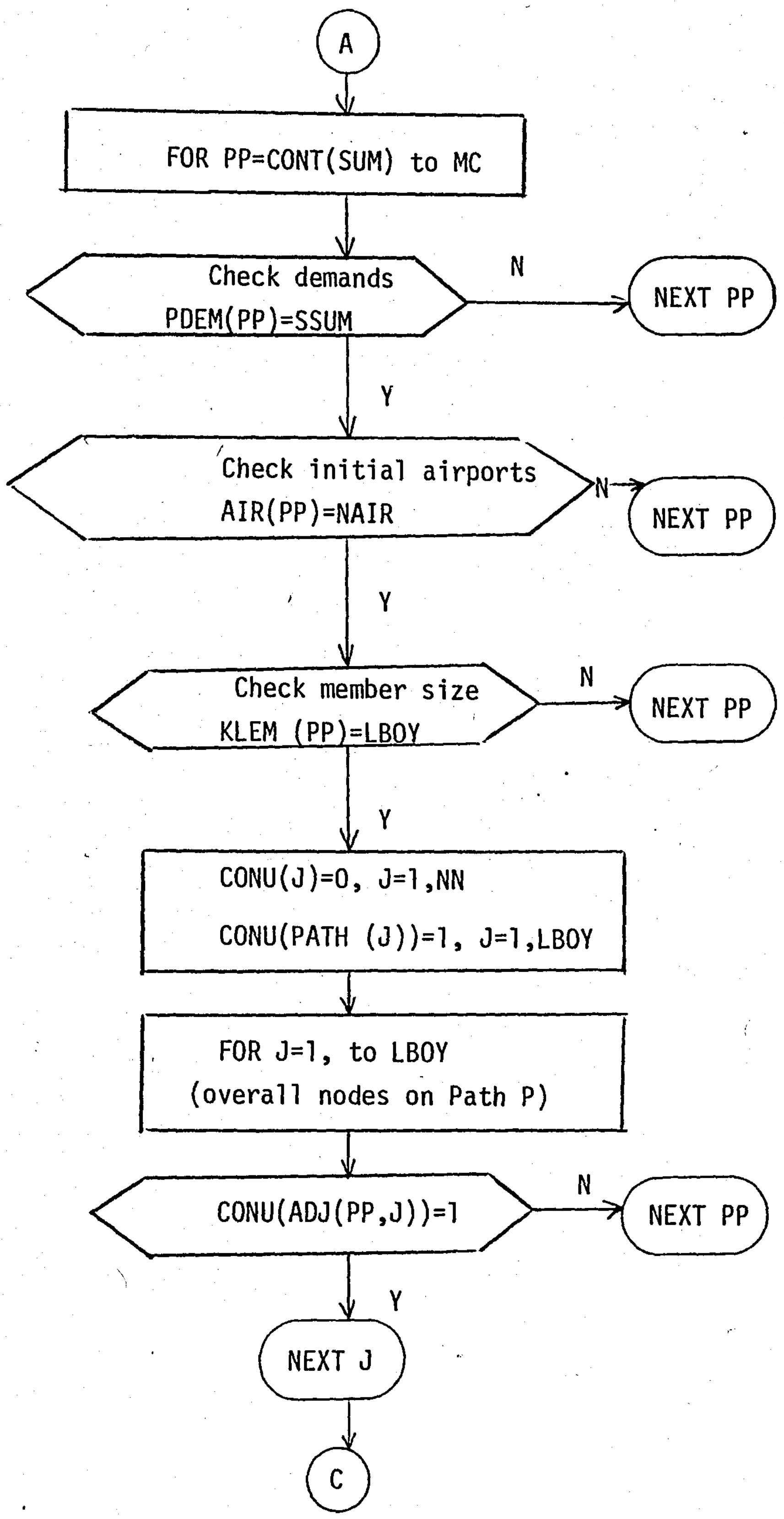
C.2 ELIMINATION

C.2.1 Definitions

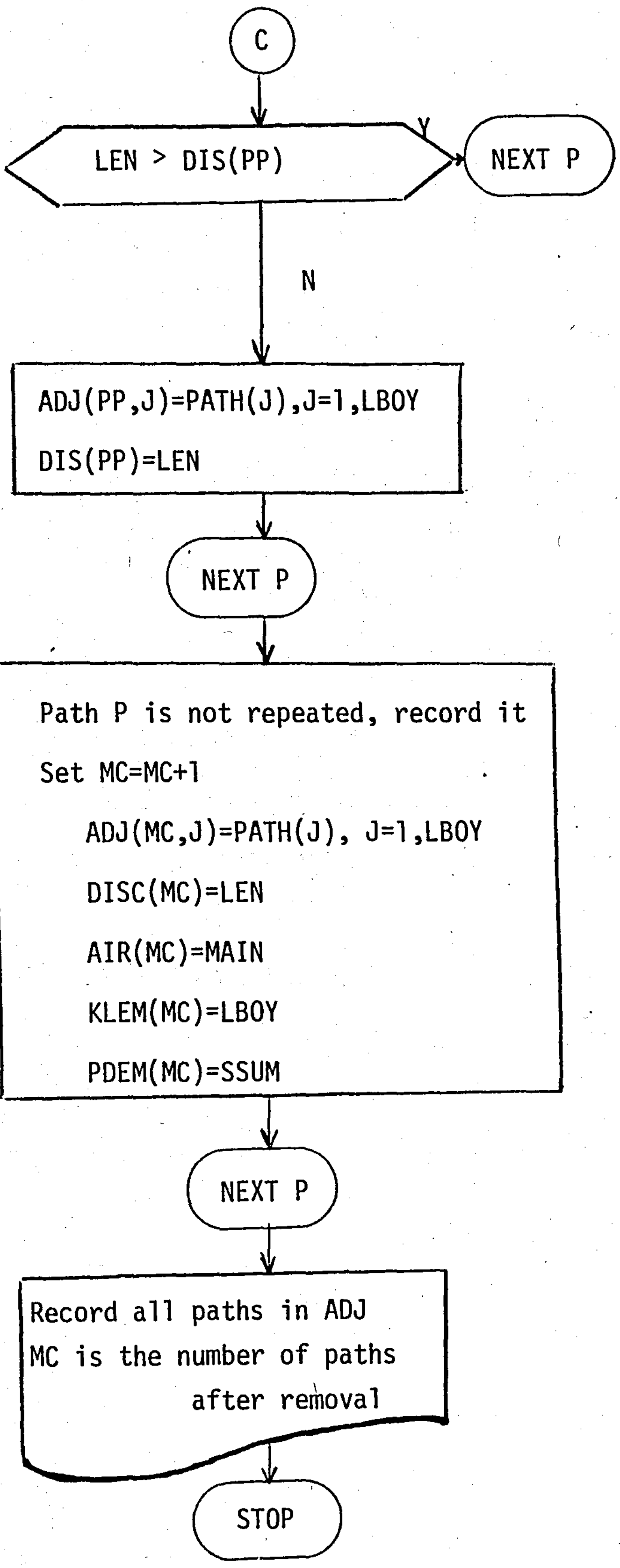
a)	MC	: Counter of paths eliminated.
b)	PATH	: Path currently read.
c)	LBOY	: Number of elements on current path (the load cardinality).
d)	LEN	: Length of current path.
e)	NAIR	: Initial airport of current path.
f)	SUM	: Demand of current path.
g)	TDMAX	: Maximum path demand.
h)	ADJ	: Array containing paths which are eliminated and stored in memory.
i)	KLEM	: Array containing number of elements of paths in ADJ.
j)	DIS	: Array containing lengths of path in ADJ.
k)	AIR	: Array containing the initial airports of paths in ADJ.
l)	PDEM	: Array containing demand of paths in ADJ.
m)	CONT	: Occurrence of path demand control array.

C.2.2 Flowchart 2. Elimination of Paths


MC = 0
FOR P = 1 to KTH
(overall simple paths generated)
Get Path P from mass storage
READ; LBOY, LEN, NAIR, (PATH(J), J=1,LBOY)
Calculate demand of Path
SUM: Sum of demands of nodes on path P and demand of airport
SSUM = SUM
SUM > TDMAX
Y
SUM = TDMAX
CONT(SUM)= 0
Y
CONT(SUM)= MC+1
N
A



A
FOR PP=CONT(SUM) to MC
Check demands
PDEM(PP)=SSUM
N
NEXT PP
Y
Check initial airports
AIR(PP)=NAIR
N
NEXT PP
Y
Check member size
KLEM (PP)=LBOY
N
NEXT PP
Y
CONU(J)=0, J=1,NN
CONU(PATH (J))=1, J=1,LBOY
FOR J=1, to LBOY
(overall nodes on Path P)
CONU(ADJ(PP,J))=1
N
NEXT PP
Y
NEXT J
C



```
ADJ(PP,J)=PATH(J),J=1,LBOY
DIS(PP)=LEN
```

```
Path P is not repeated, record it
Set MC=MC+1
    ADJ(MC,J)=PATH(J), J=1,LBOY
    DISC(MC)=LEN
    AIR(MC)=MAIN
    KLEM(MC)=LBOY
    PDEM(MC)=SSUM
```

```
Record all paths in ADJ
MC is the number of paths
            after removal
```

C.3 BRP ALGORITHM

C.3.1 Definitions

a)	R	: the number of airports.
b)	M	: the number of loads.
c)	P	: the number of planes.
d)	RP	: the number of airports with planes initially.
e)	MAIR	: Plane availability at airports initially MAIR(I): number of planes at airport I initially.
f)	TOTAL	: Total frequency array TOTAL(I): total frequency of load I in path list.
g)	GMAX	: Maximum number of members per path for the paths within the path list.
h)	USET	: Number of paths in the path list.
i)	LSET	: Top search limit for the first plane.
j)	KSET	: Bottom search limit.
k)	KRT	: Top search limit.
l)	KPL	: Current number of planes used in partial solution OPT.
m)	KLD	: Current number of loads covered by partial solution OPT.
o)	D	: Airport-to-airport flight time matrix. D(I,J): Time required for a non-stop flight from airport I to airport J.

The following variables and arrays are expressed in terms of the nodes in the transformed network:

a)	AIRMAX	: Plane availability at airports initially AIRMAX(I): number of planes at airport I initially.

b) AIRMED : Current plane usage from each airport.

AIRMED(I): number of planes used from airport I.

c) PATH : Ith path taken from mass storage.

d) LBOY : Number of loads on Ith path (member size).

e) LEN : Length of Ith path.

f) NAIR : Initial port of Ith path.

g) ADJ : The sorted path list.

h) KLEM : Array containing number of loads on paths within the path list.

i) DIS : Array containing length of path within the path list.

j) AIR : Array containing starting airports of paths within the path list.

k) ADRES : The address table between the path lists.

l) FRE : The frequency matrix

FRE (I,J) = number of times node J occurred on paths with load cardinality i in the path list; if J > 1.

number of disjoint loads on paths with load cardinality i within the path list; if J = 1.

m) FIRST : The first occurrence matrix.

FIRST (I,J) = Path number of first occurrence of node J occurred on paths with load cardinality i in the path list; if J > 1.

Path number of first occurrence of a path with load cardinality i within the path list; if J = 1.

n) YFIRST : Firs occurrence of members in the new list.

YFIRST(I) = first occurrence path number of a path with load cardinality i in the new list.

o) SFIRST : First occurrence of loads in the new list.

SFIRST(I) = First occurrence path number of load I in the new list.

p) COVER : Indicates whether loads are covered by current partial solution or not.

COVER(I) = 1 ; if load I is covered.

0 ; otherwise.

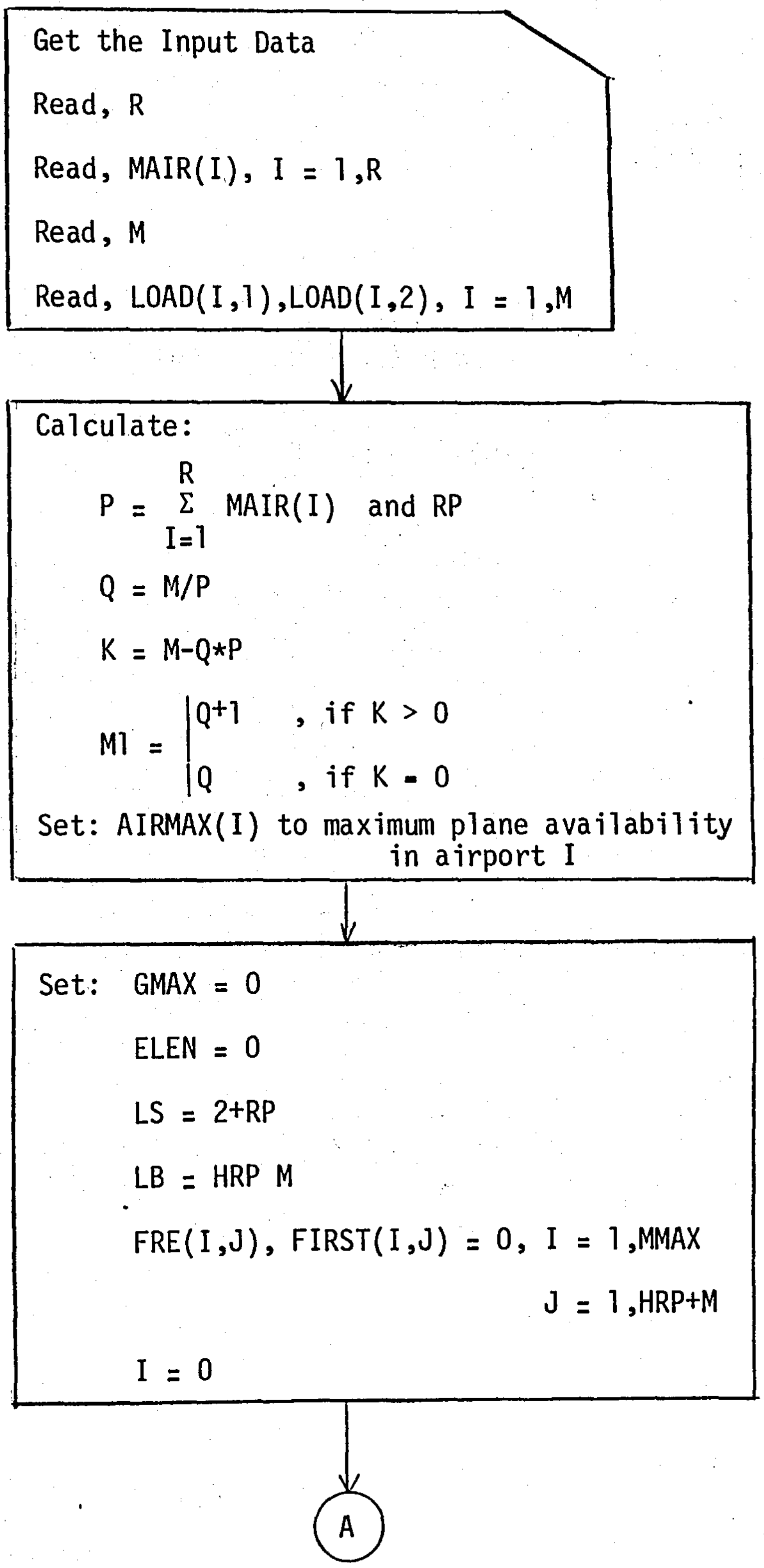
q) OPT : Current partial solution.

OPT(I) : shows the number of paths in old list assigned to Ith plane in current partial solution.

r) I membered path J: The path J (i.e. set S_J) which covers I loads (I rows).

s) node J : The meaning of nodes are kept the same as in the network formulation in Section 3.1. Except, source node (node 1) and terminal node (node T) have no meaning furthermore.

C.3.2 Flowchart 3



Get the Input Data

Read, R

Read, MAIR(I), I = 1,R

Read, M

Read, LOAD(I,1),LOAD(I,2), I = 1,M

Calculate:

$$P = \sum_{I=1}^{R} MAIR(I) \text{ and } RP$$

$$Q = M/P$$

$$K = M-Q*P$$

$$M1 = \begin{cases} Q+1 & \text{, if } K > 0 \\ Q & \text{, if } K = 0 \end{cases}$$

Set: AIRMAX(I) to maximum plane availability in airport I

Set: GMAX = 0

ELEN = 0

LS = 2+RP

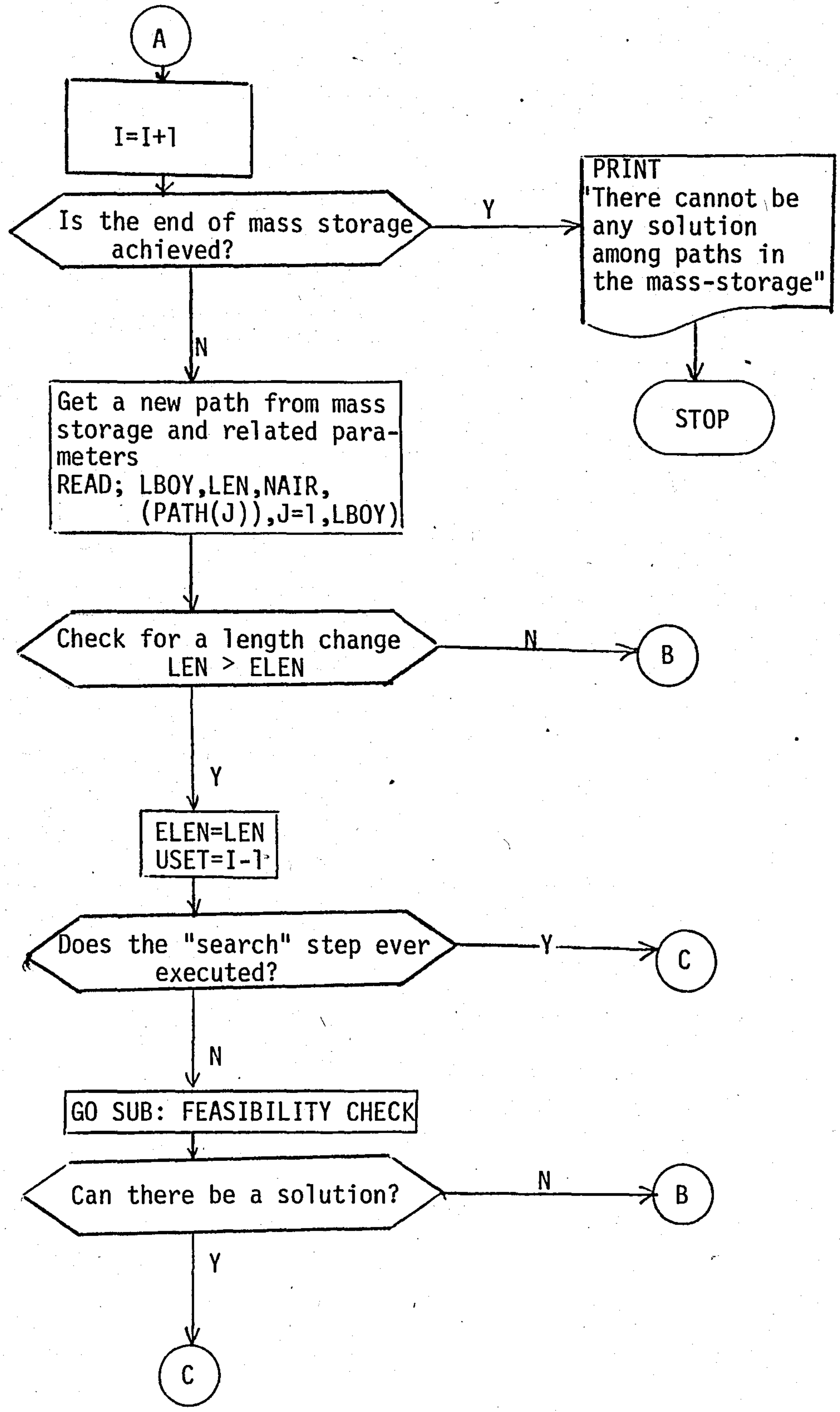
LB = HRP M

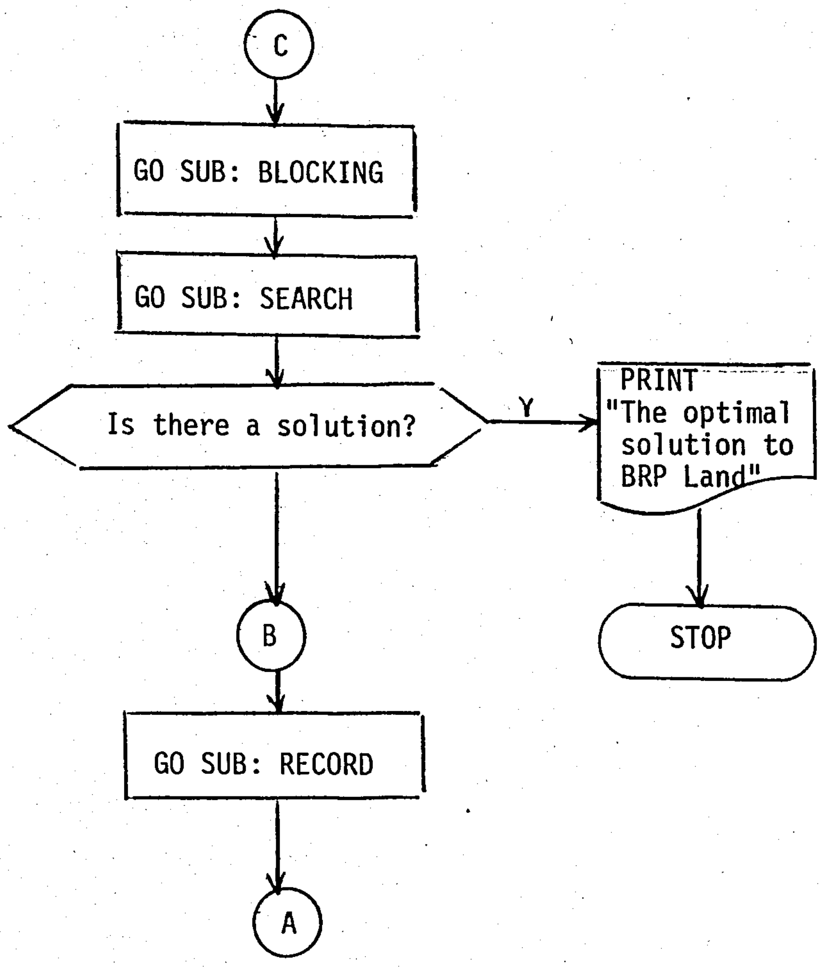
FRE(I,J), FIRST(I,J) = 0, I = 1,MMAX

J = 1,HRP+M

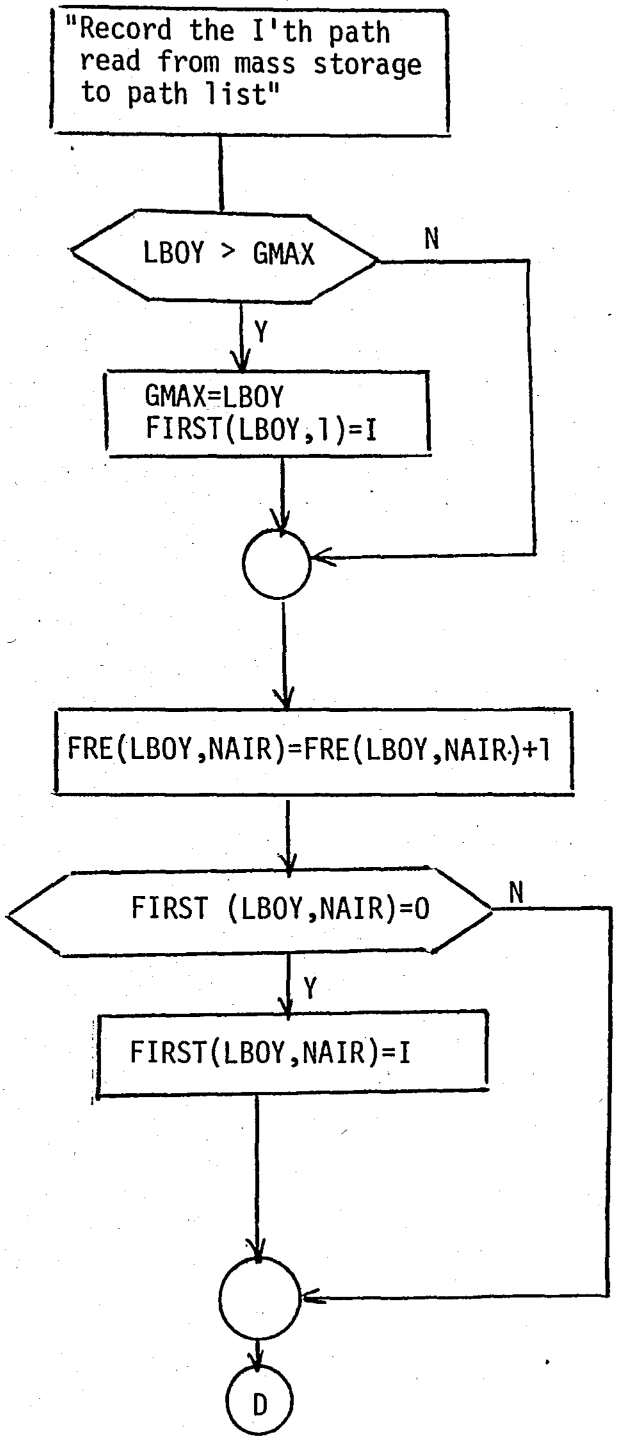
I = 0

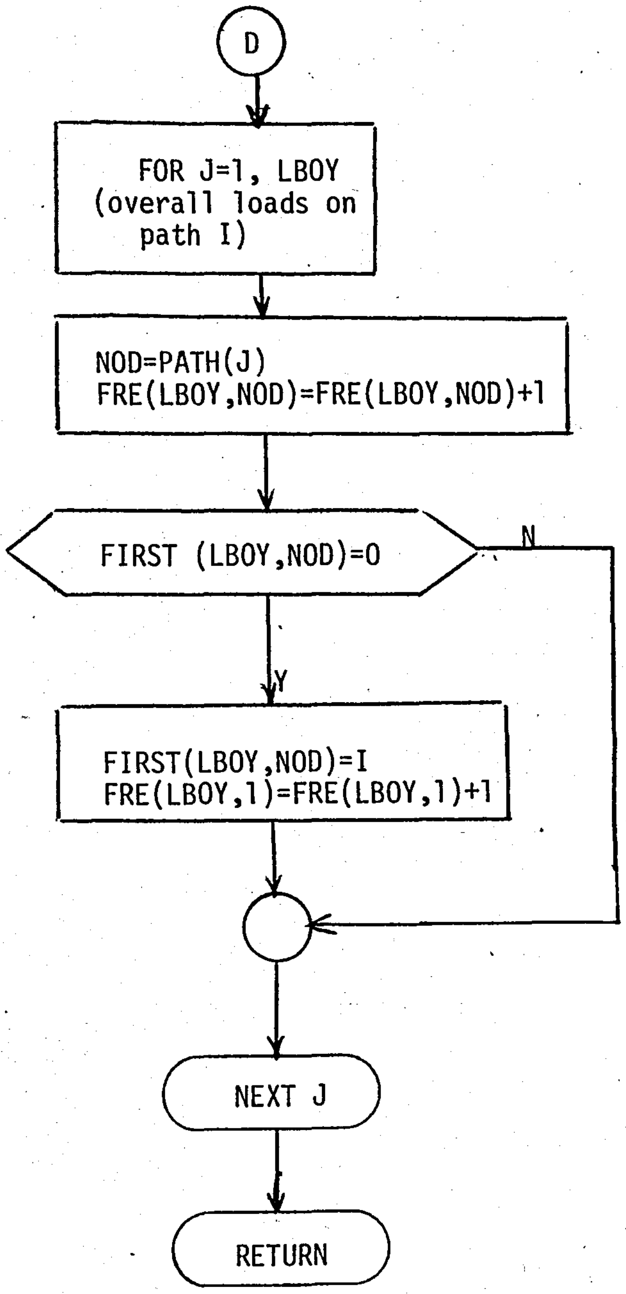
A


A
I=I+1
Is the end of mass storage achieved?
Y
PRINT "There cannot be any solution among paths in the mass-storage"
STOP
N
Get a new path from mass storage and related parameters
READ; LBOY,LEN,NAIR, (PATH(J)),J=1,LBOY)
Check for a length change LEN > ELEN
N
B
Y
ELEN=LEN
USET=I-1
Does the "search" step ever executed?
Y
C
N
GO SUB: FEASIBILITY CHECK
Can there be a solution?
N
B
Y
C


C
GO SUB: BLOCKING
GO SUB: SEARCH
Is there a solution?
Y
PRINT "The optimal solution to BRP Land"
STOP
B
GO SUB: RECORD
A

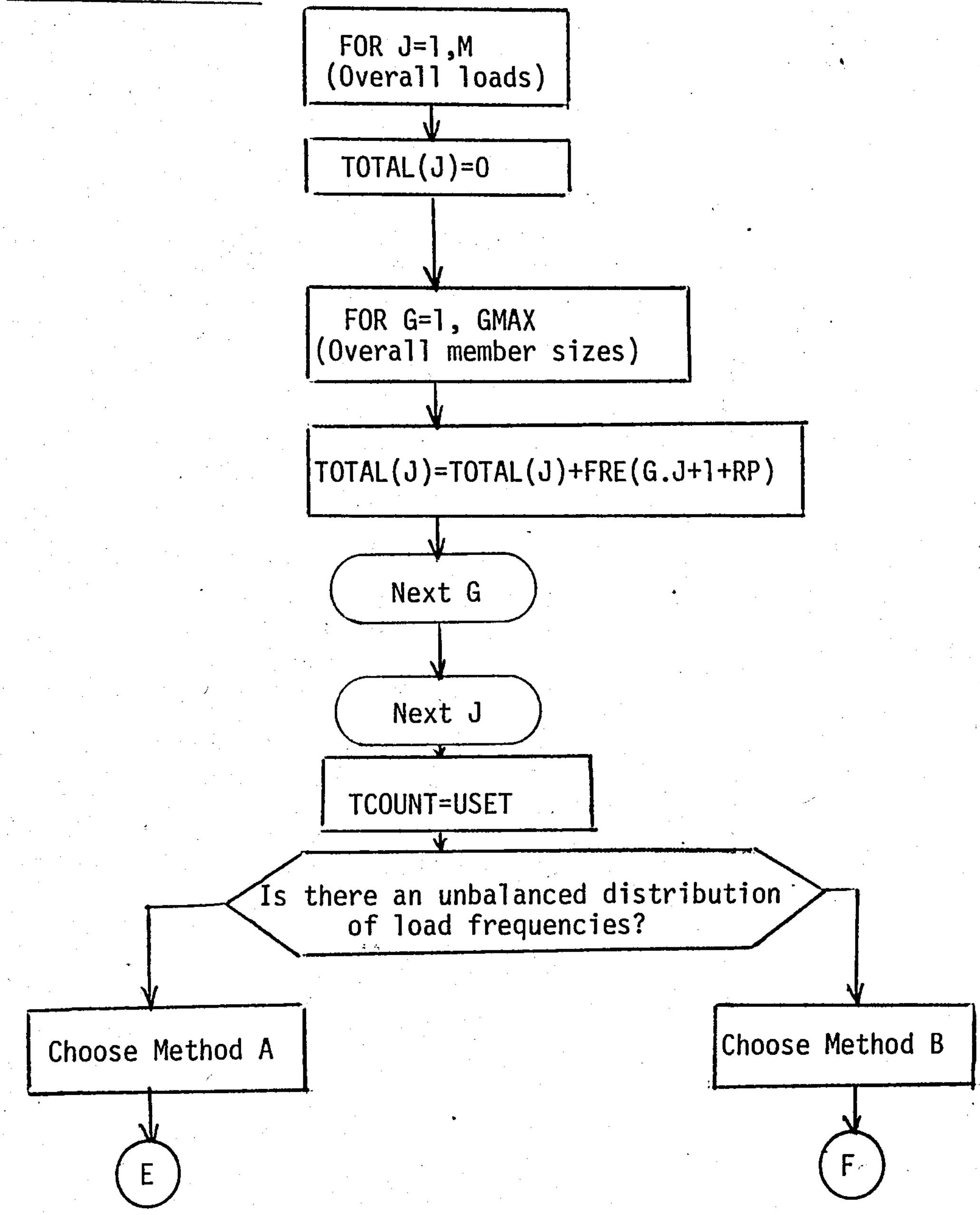
SUBROUTINE: RECORD


"Record the I'th path read from mass storage to path list"
LBOY > GMAX
N
Y
GMAX=LBOY
FIRST(LBOY,1)=I
FRE(LBOY,NAIR)=FRE(LBOY,NAIR)+1
FIRST (LBOY,NAIR)=0
N
Y
FIRST(LBOY,NAIR)=I
D

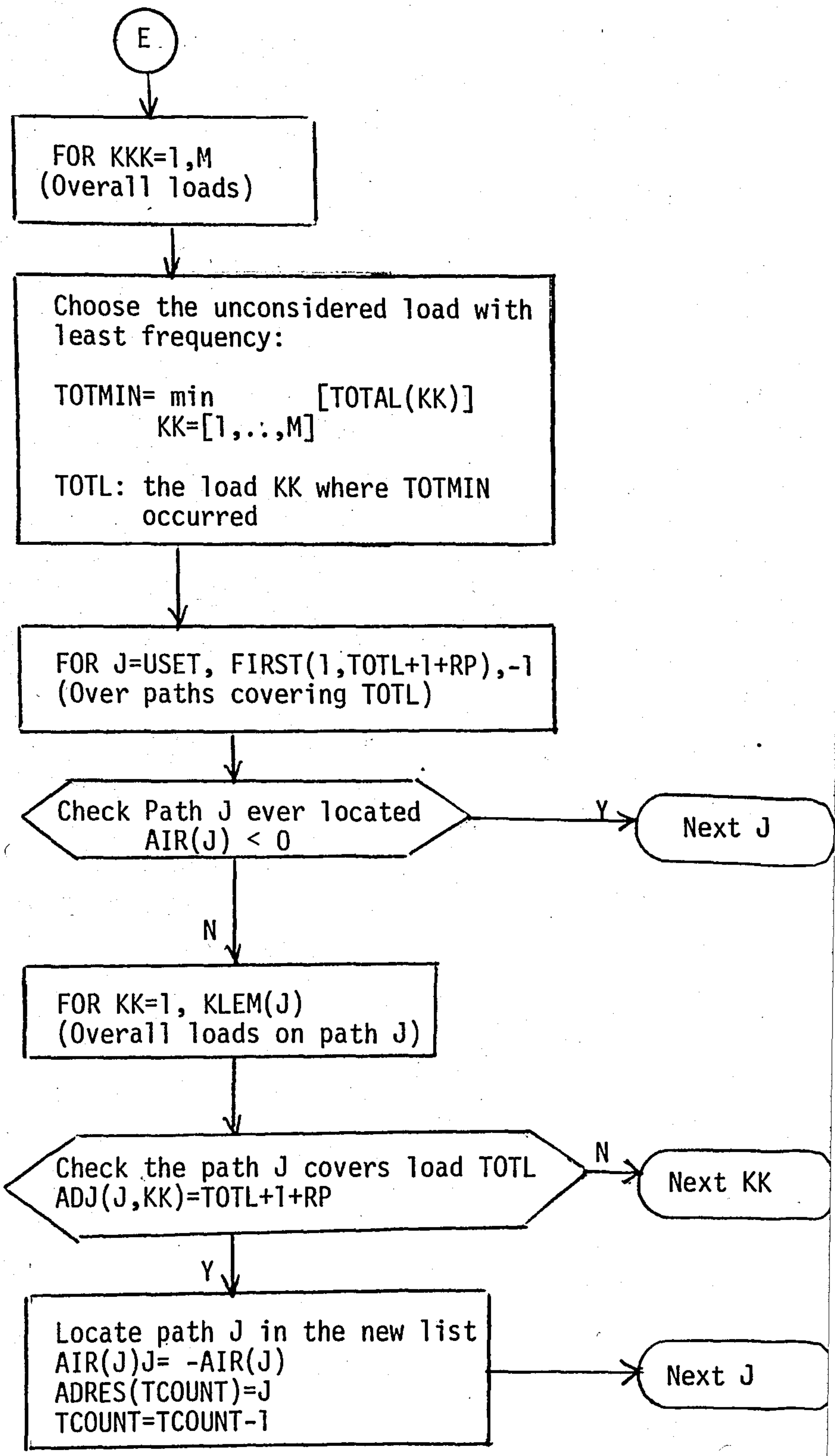

D
FOR J=1, LBOY (overall loads on path I)
NOD=PATH(J)
FRE(LBOY,NOD)=FRE(LBOY,NOD)+1
FIRST (LBOY,NOD)=0
N
Y
FIRST(LBOY,NOD)=I
FRE(LBOY,1)=FRE(LBOY,1)+1
NEXT J
RETURN

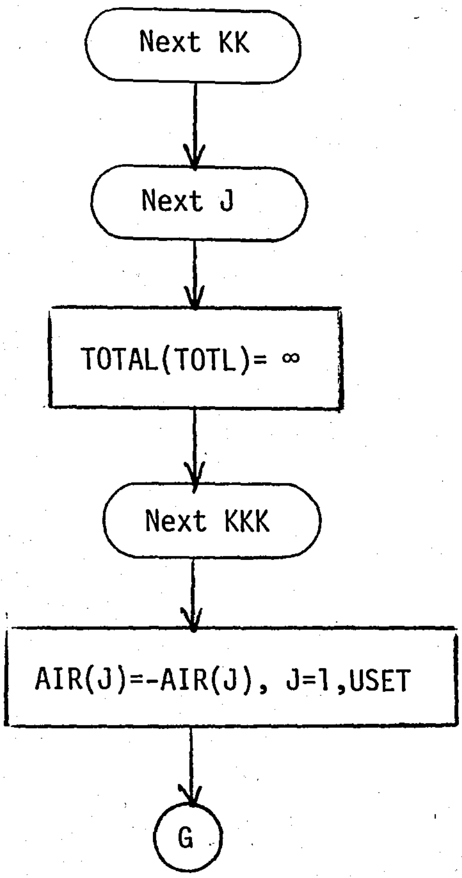
SUBROUTINE: BLOCK

Calculate Total Load Frequencies:

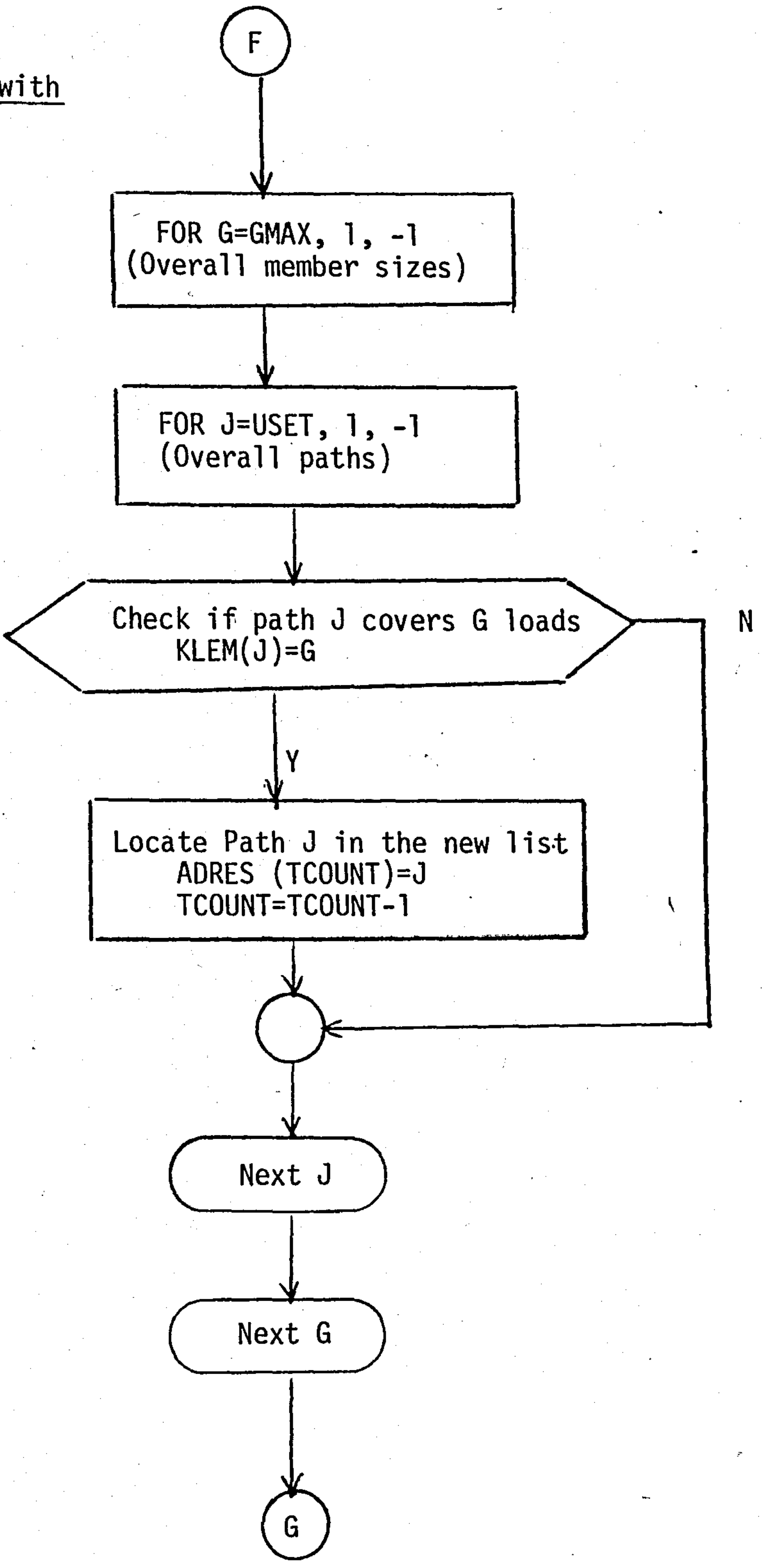


Blocking with Method A:

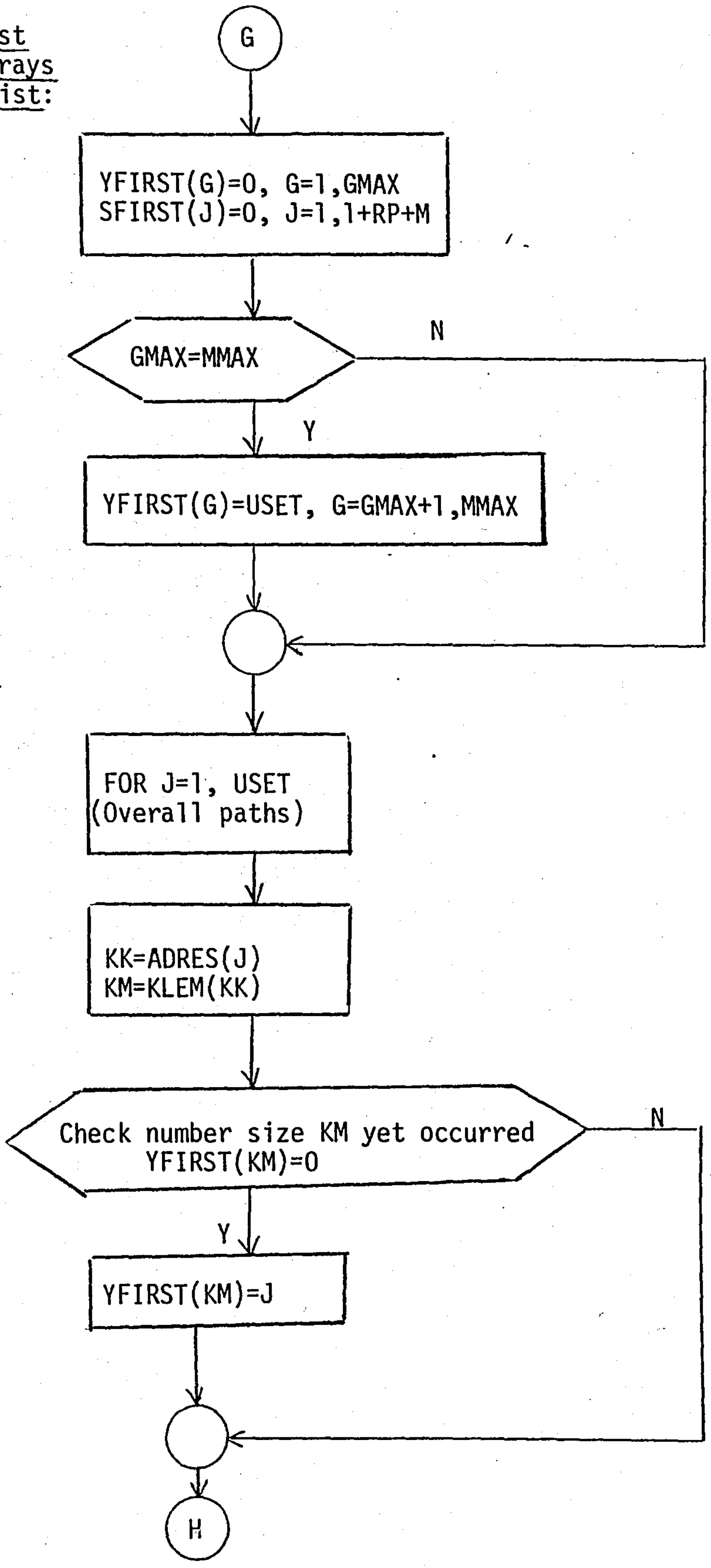

E
FOR KKK=1,M (Overall loads)
Choose the unconsidered load with least frequency:
TOTMIN= min [TOTAL(KK)] KK=[1,..,M]
TOTL: the load KK where TOTMIN occurred
FOR J=USET, FIRST(1,TOTL+1+RP),-1 (Over paths covering TOTL)
Check Path J ever located AIR(J) < 0
Y
Next J
N
FOR KK=1, KLEM(J) (Overall loads on path J)
Check the path J covers load TOTL ADJ(J,KK)=TOTL+1+RP
N
Next KK
Y
Locate path J in the new list AIR(J)J= -AIR(J) ADRES(TCOUNT)=J TCOUNT=TCOUNT-1
Next J

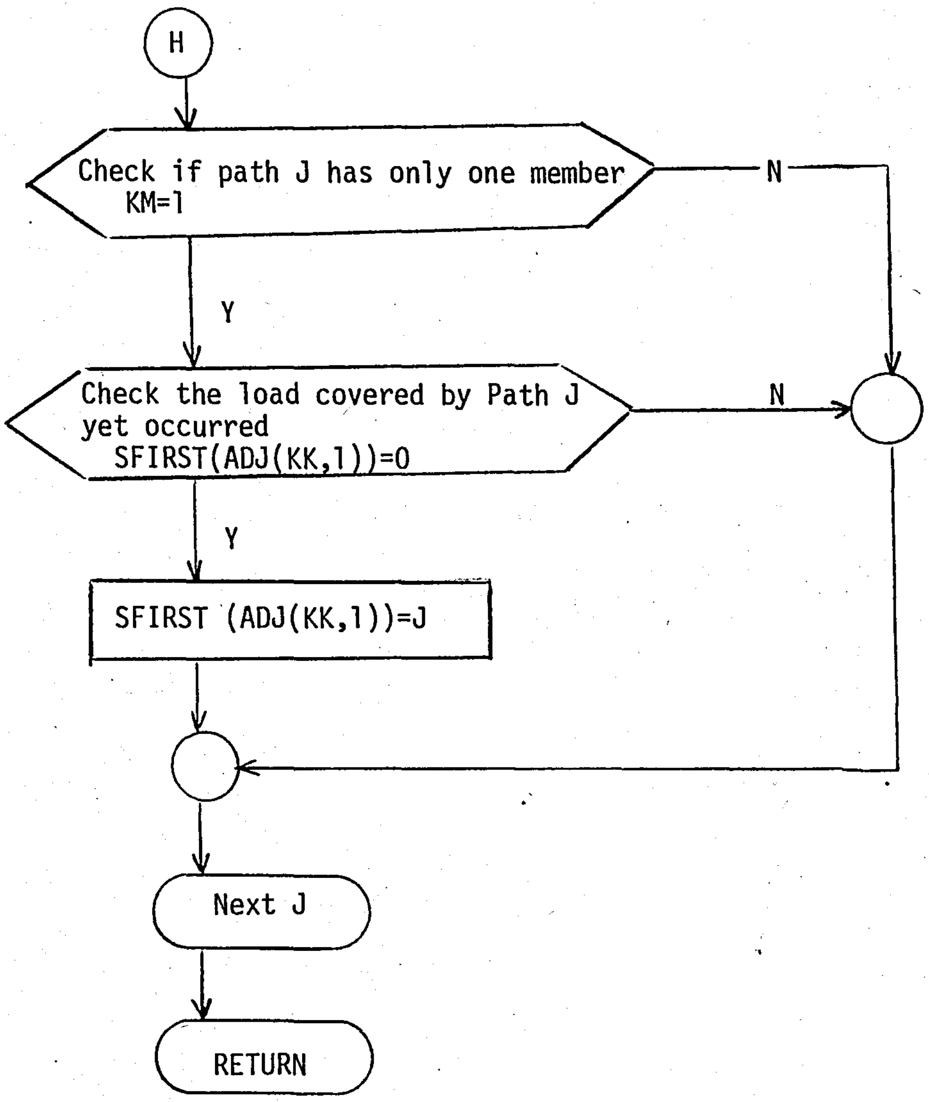

Next KK
Next J
TOTAL(TOTL)= ∞
Next KKK
AIR(J)=-AIR(J), J=1,USET
G

<u>Blocking with Method B:</u>



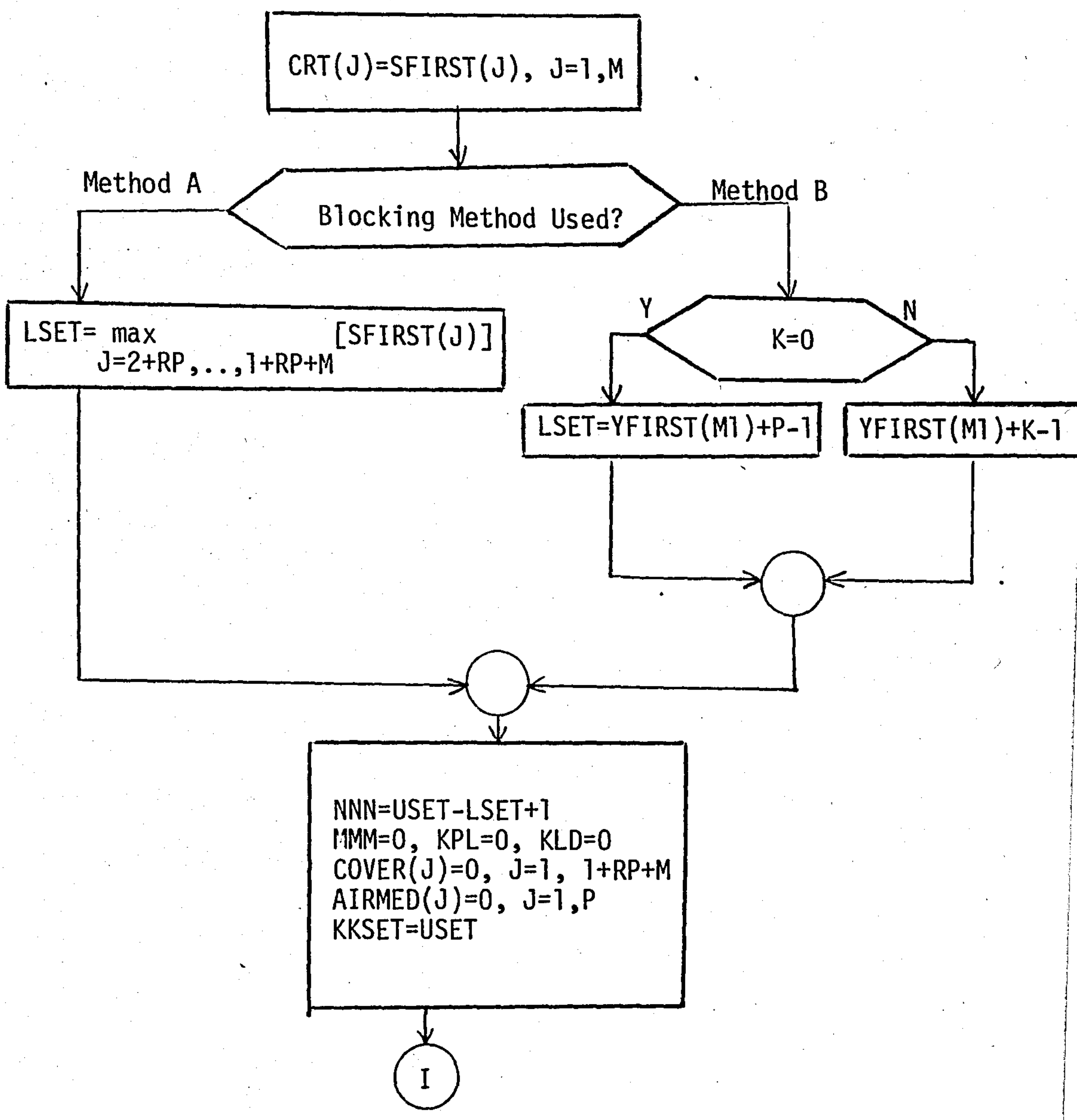
Calculate First Occurrance Arrays on new Path List:



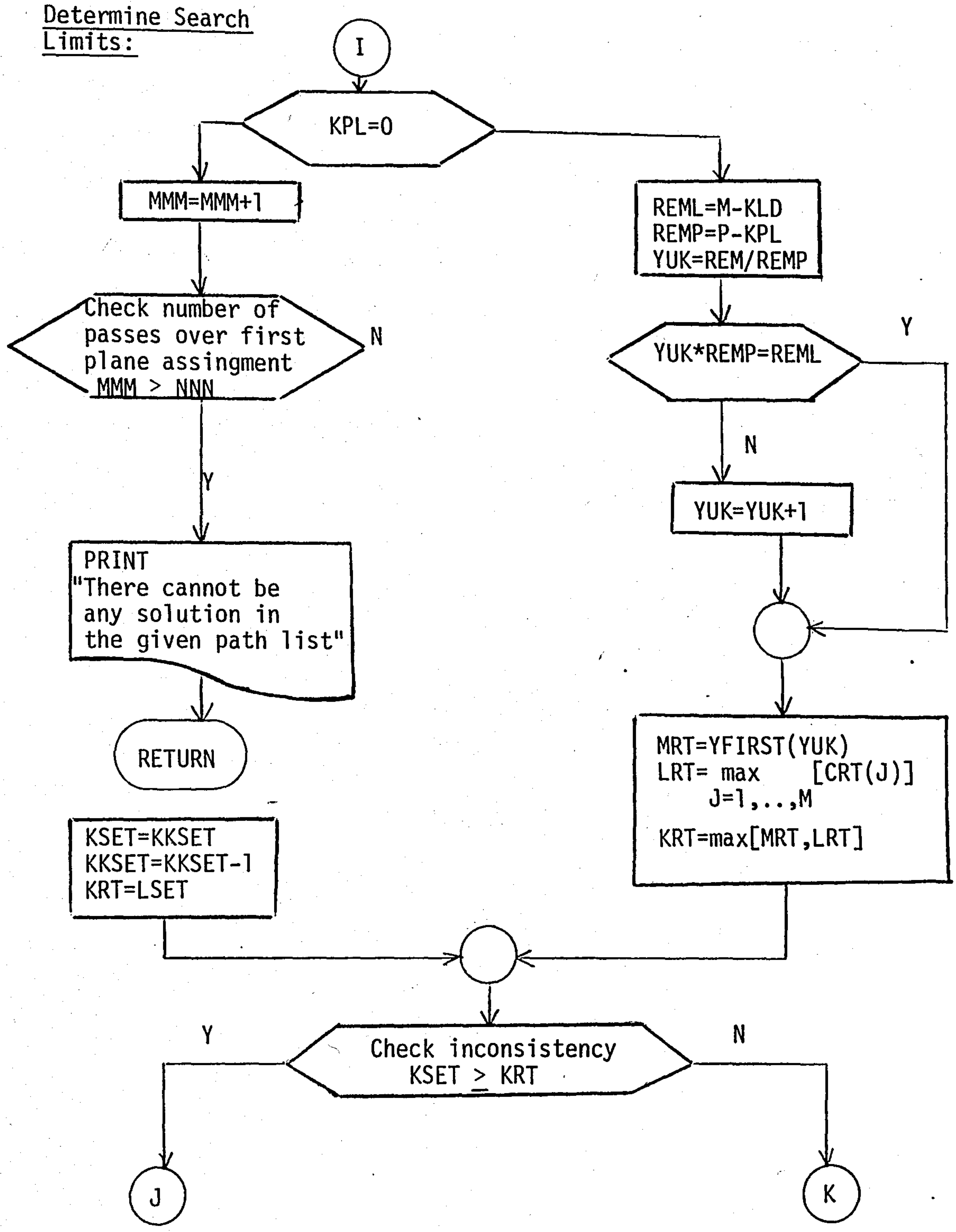

H
Check if path J has only one member
KM=1
N
Y
Check the load covered by Path J
yet occurred
SFIRST(ADJ(KK,1))=0
N
Y
SFIRST (ADJ(KK,1))=J
Next J
RETURN

SUBROUTINE: SEARCH

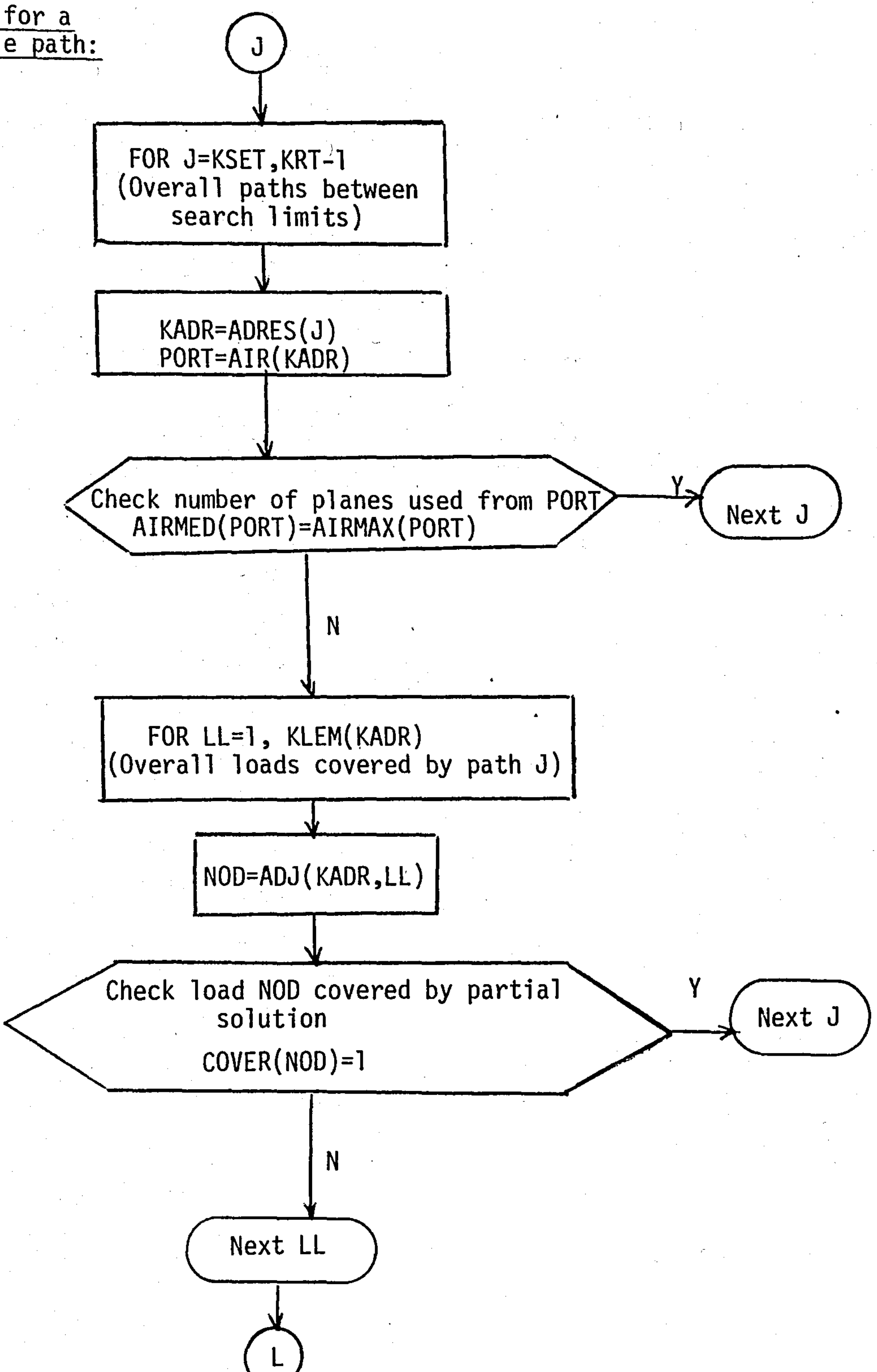
Initialize for Search:



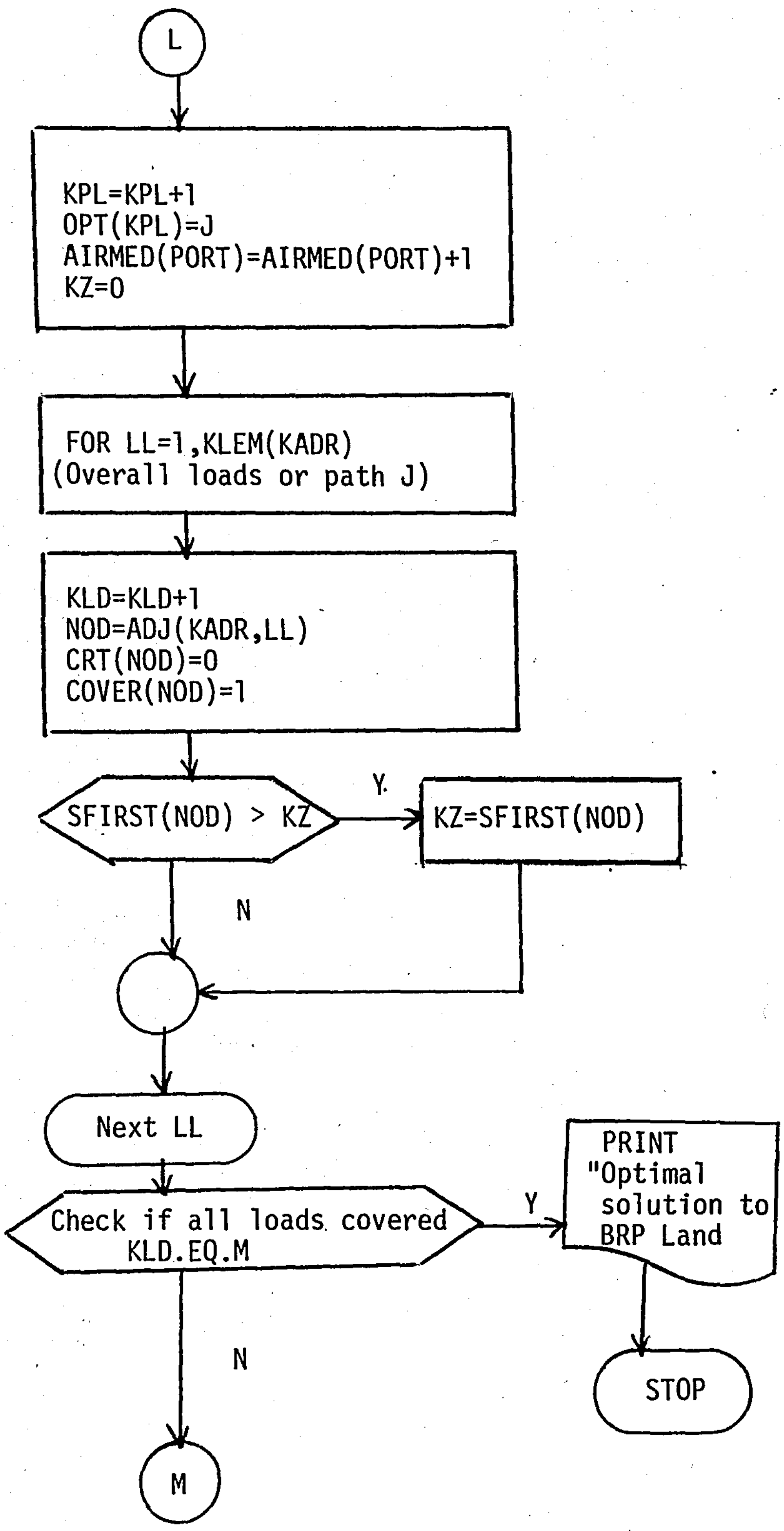
Determine Search Limits:

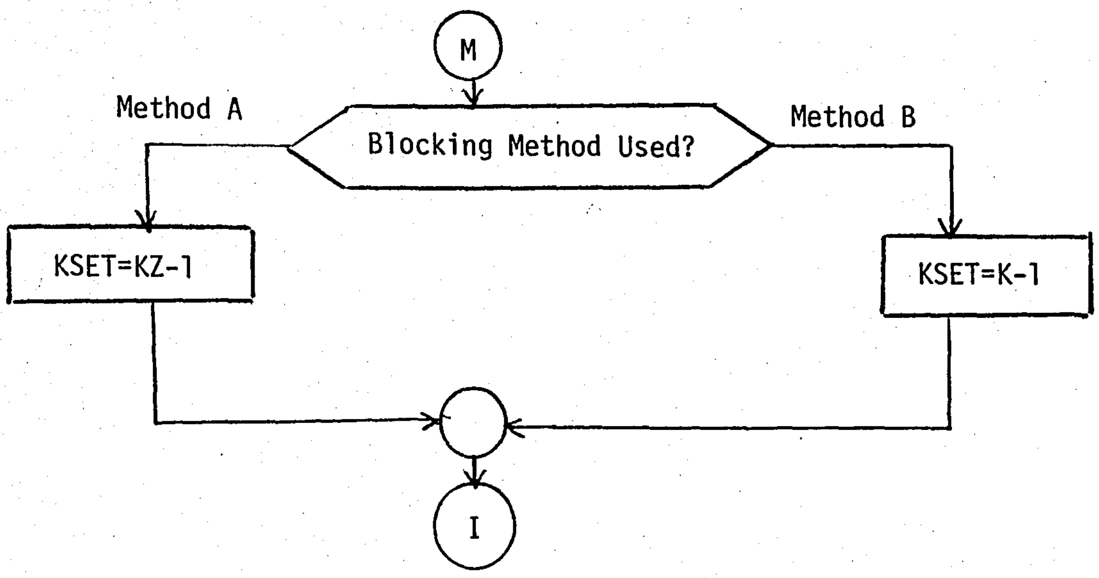

I
KPL=0
MMM=MMM+1
Check number of passes over first plane assingment MMM > NNN
N
Y
PRINT "There cannot be any solution in the given path list"
RETURN
KSET=KKSET
KKSET=KKSET-1
KRT=LSET
REML=M-KLD
REMP=P-KPL
YUK=REM/REMP
YUK*REMP=REML
Y
N
YUK=YUK+1
MRT=YFIRST(YUK)
LRT= max [CRT(J)] J=1,..,M
KRT=max[MRT,LRT]
Check inconsistency KSET ≥ KRT
Y
N
J
K

Search for a Feasible path:

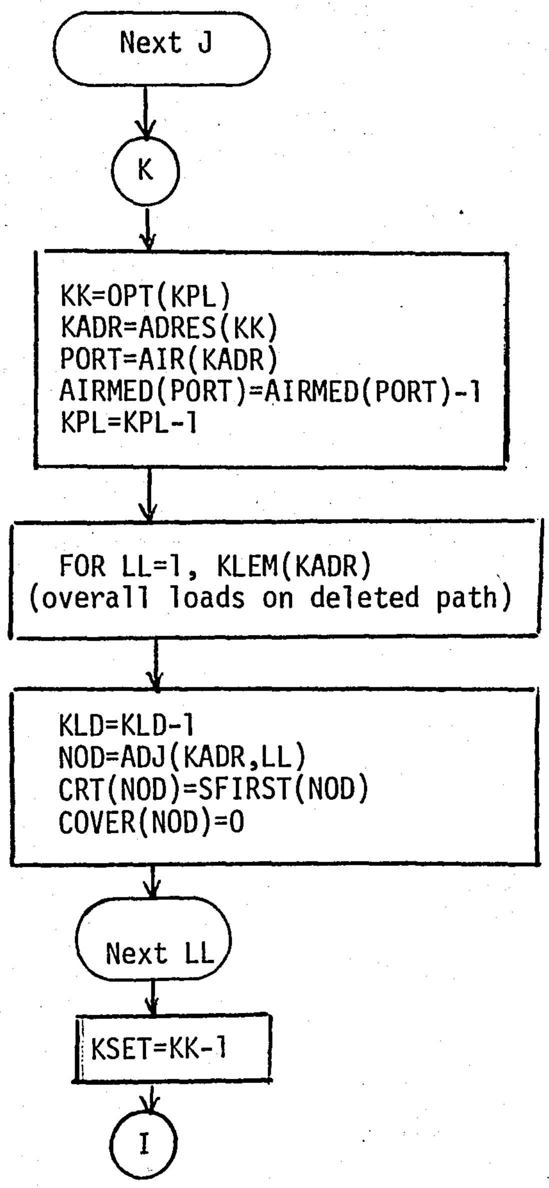

J
FOR J=KSET,KRT-1
(Overall paths between search limits)
KADR=ADRES(J)
PORT=AIR(KADR)
Check number of planes used from PORT
AIRMED(PORT)=AIRMAX(PORT)
Y
Next J
N
FOR LL=1, KLEM(KADR)
(Overall loads covered by path J)
NOD=ADJ(KADR,LL)
Check load NOD covered by partial solution
COVER(NOD)=1
Y
Next J
N
Next LL
L

Include path J to partial solution:

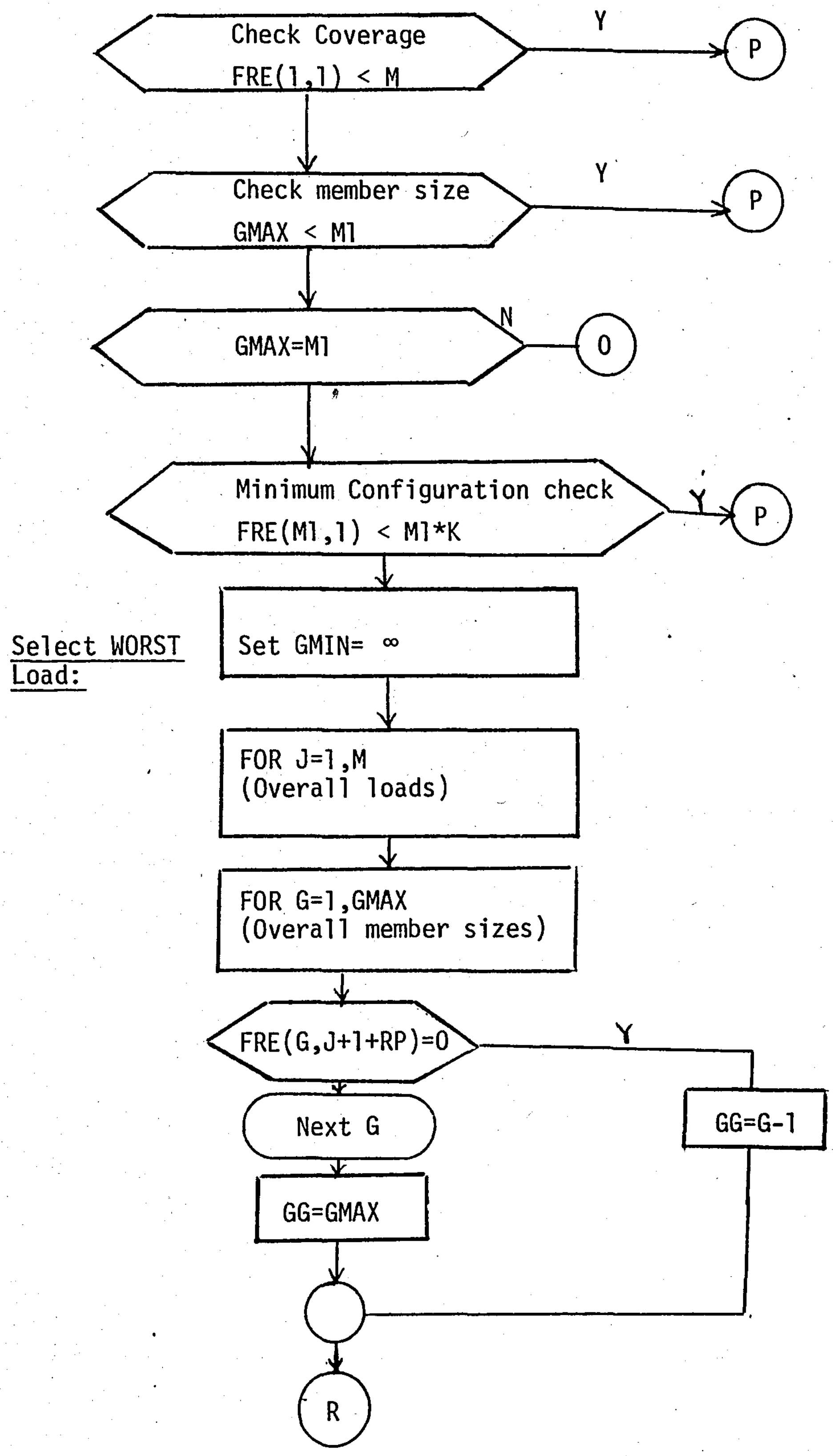


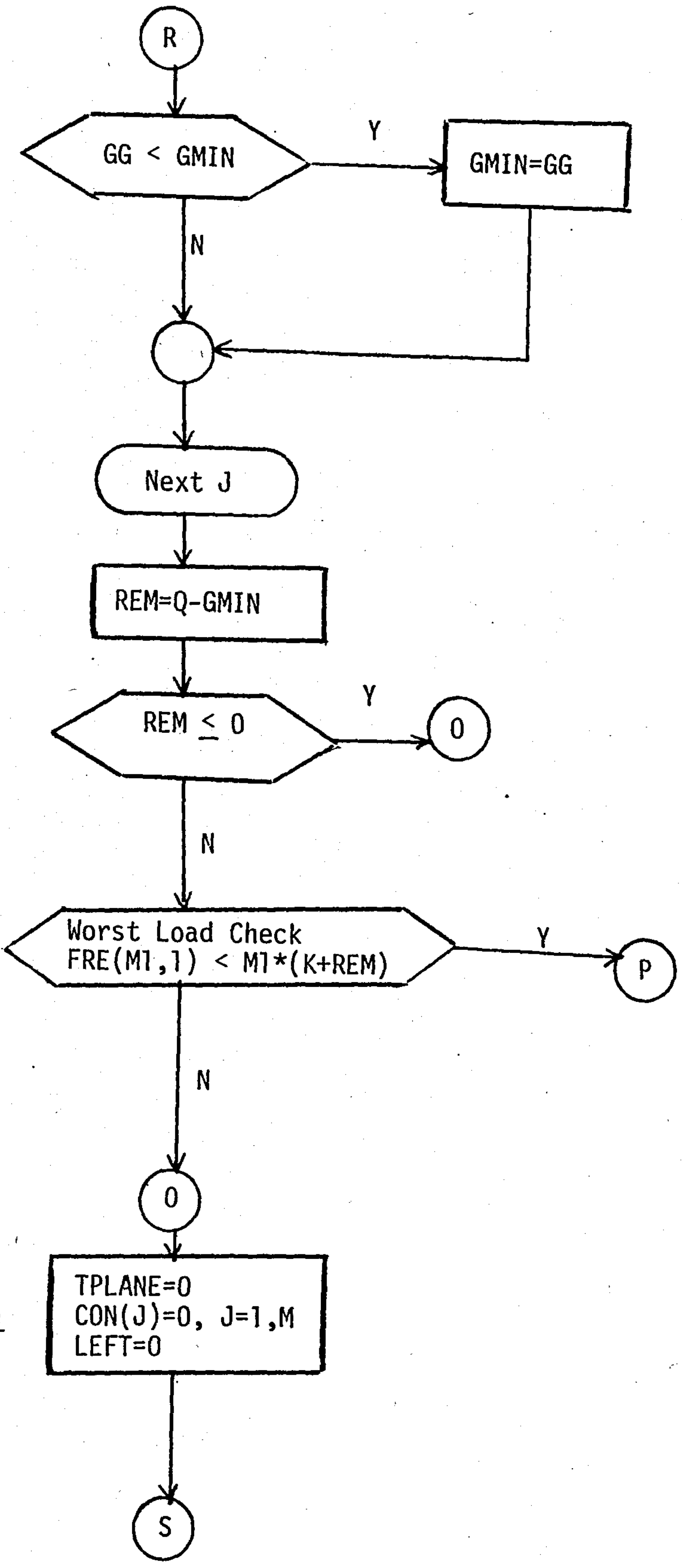


Delete the last path added to partial solution:

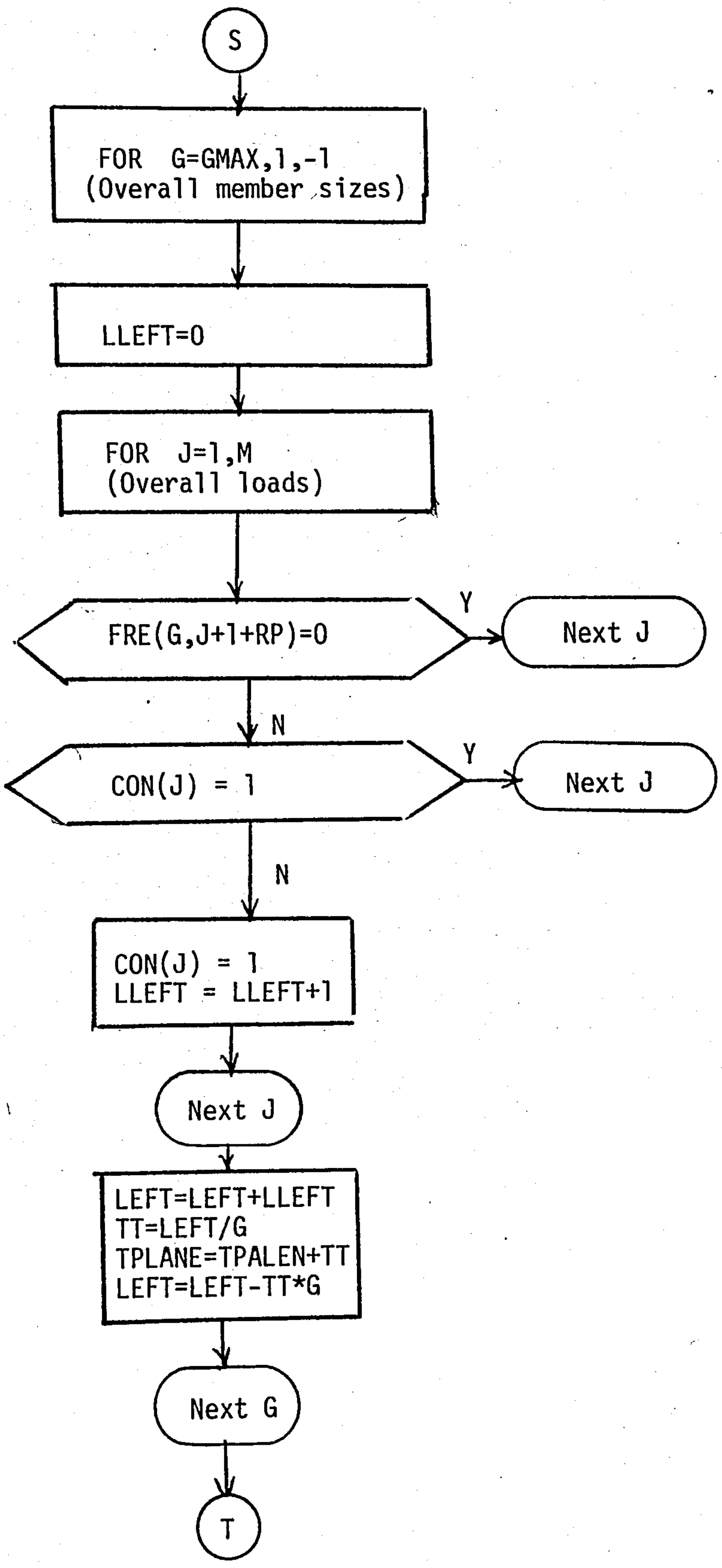


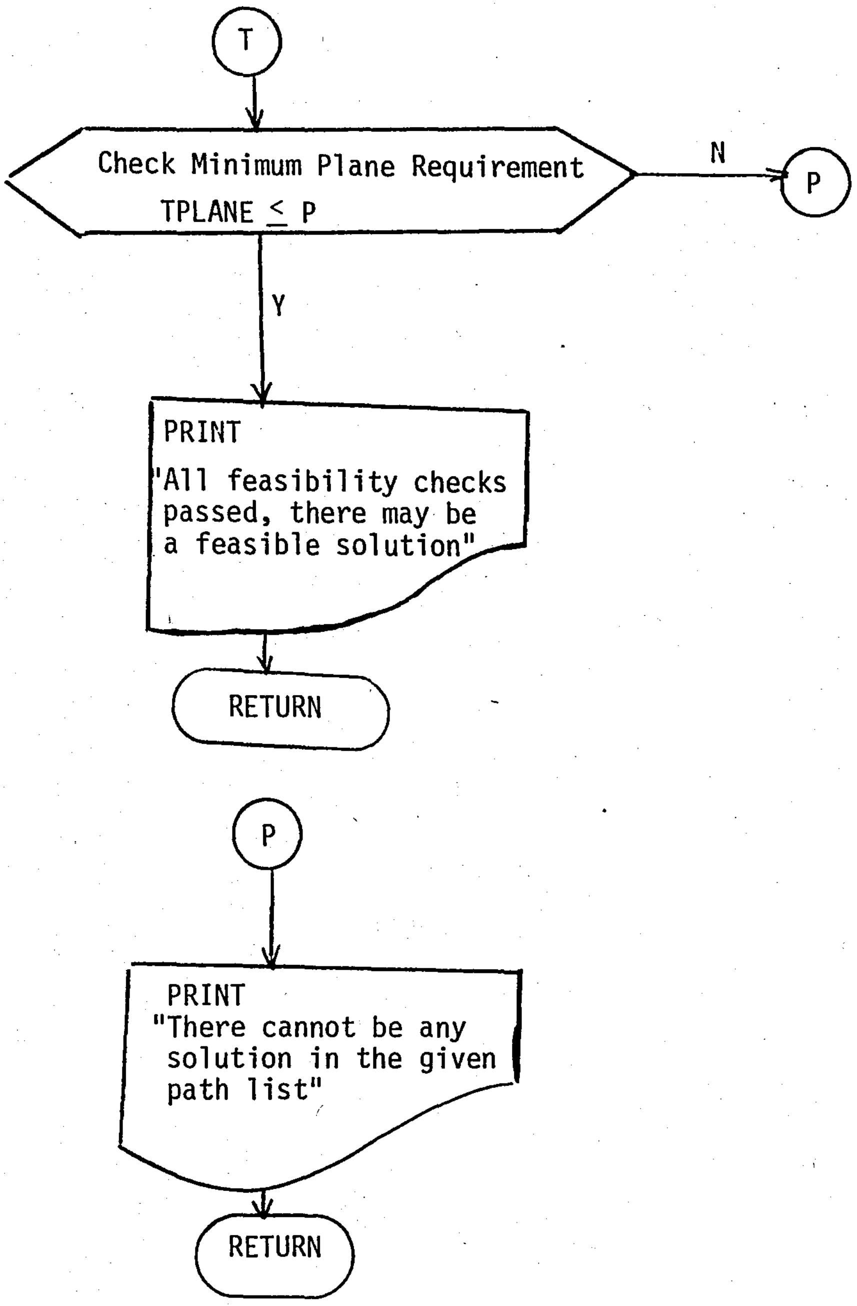
SUBROUTINE: FEASIBILITY CHECK





Calculate minimum plane requirement:


S
FOR G=GMAX,1,-1
(Overall member sizes)
LLEFT=0
FOR J=1,M
(Overall loads)
FRE(G,J+1+RP)=0
Y
Next J
N
CON(J) = 1
Y
Next J
N
CON(J) = 1
LLEFT = LLEFT+1
Next J
LEFT=LEFT+LLEFT
TT=LEFT/G
TPLANE=TPALEN+TT
LEFT=LEFT-TT*G
Next G
T


T
Check Minimum Plane Requirement
TPLANE $\leq$ P
N
P
Y
PRINT
"All feasibility checks passed, there may be a feasible solution"
RETURN
P
PRINT
"There cannot be any solution in the given path list"
RETURN

C.4 SCHEDULING

C.4.1 Definitions

a) USE(I) : number of planes serviced at Ith airport at given time.

b) QUSE(I) : number of planes waiting at Ith airport at given time.

c) QUE(I,J) : indicating Jth plane waiting at Ith airport.

d) COMPT(I) : completion time of current job of plane I.

e) POINT(I) : indicates the sequence number of load that plane I currently deals.

f) STATUS(I) : status of plane I.

= 1; if plane I flying empty.

= 2; if plane I flying full.

= 3; if plane I loading.

= 4; if plane I unloading.

= 5; if plane I waiting for loading.

= 6; if plane I waiting for unloading.

= 7; if plane I finished the job.

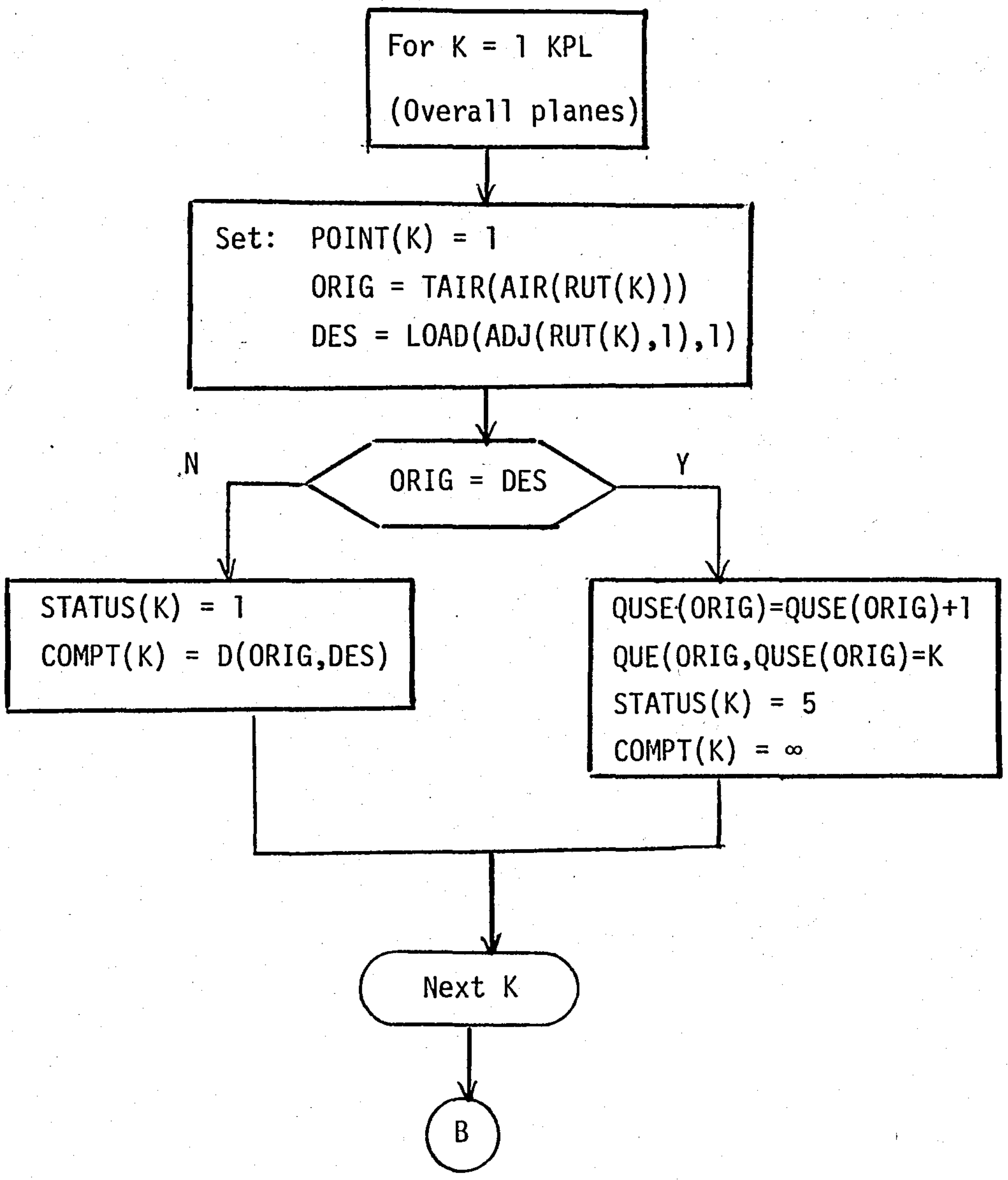
g) KPL : number of planes used in given BRP.

h) RUT(I) : indicates the path with Ith plane assigned.

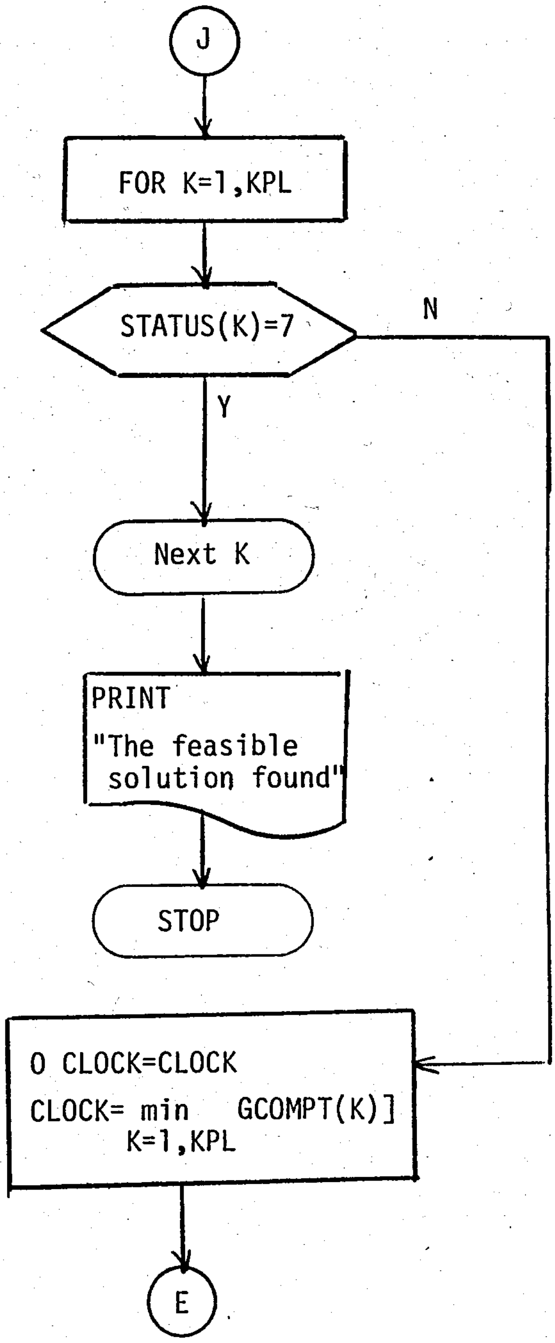
i) TAIR : array to transform meaning of airports.

C.4.2 Flowchart 4

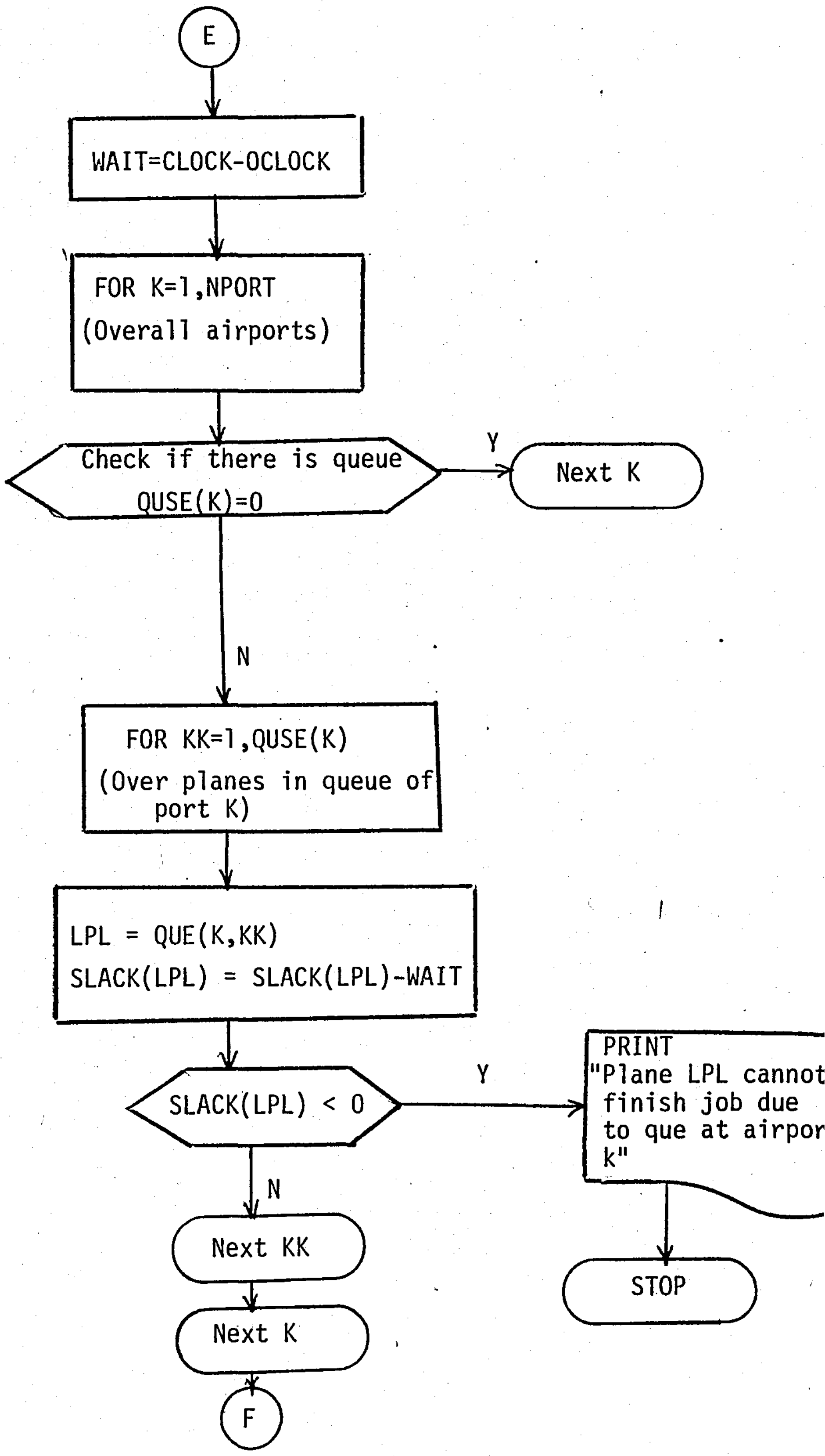
INITIALIZE



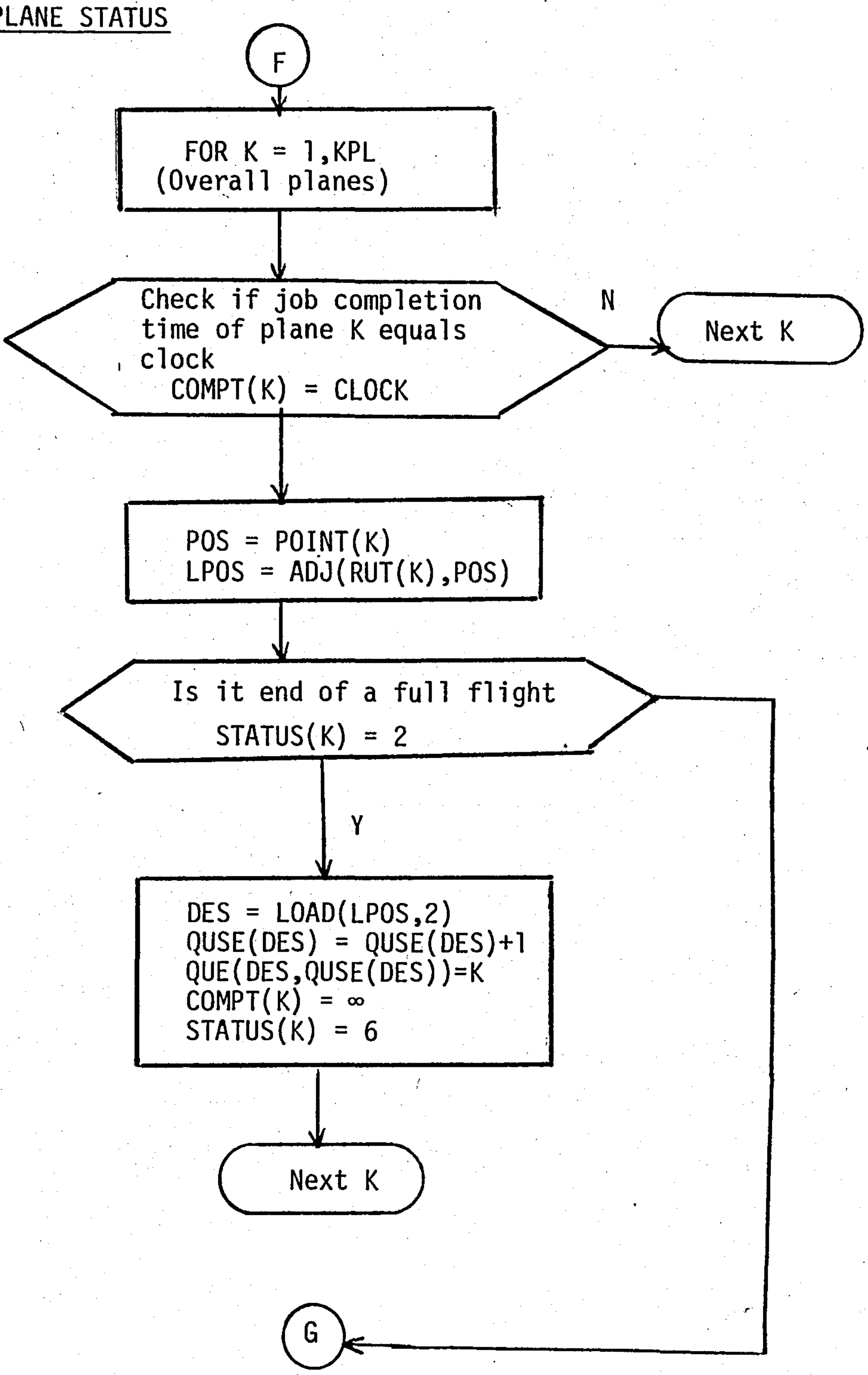
UPDATE CLOCK

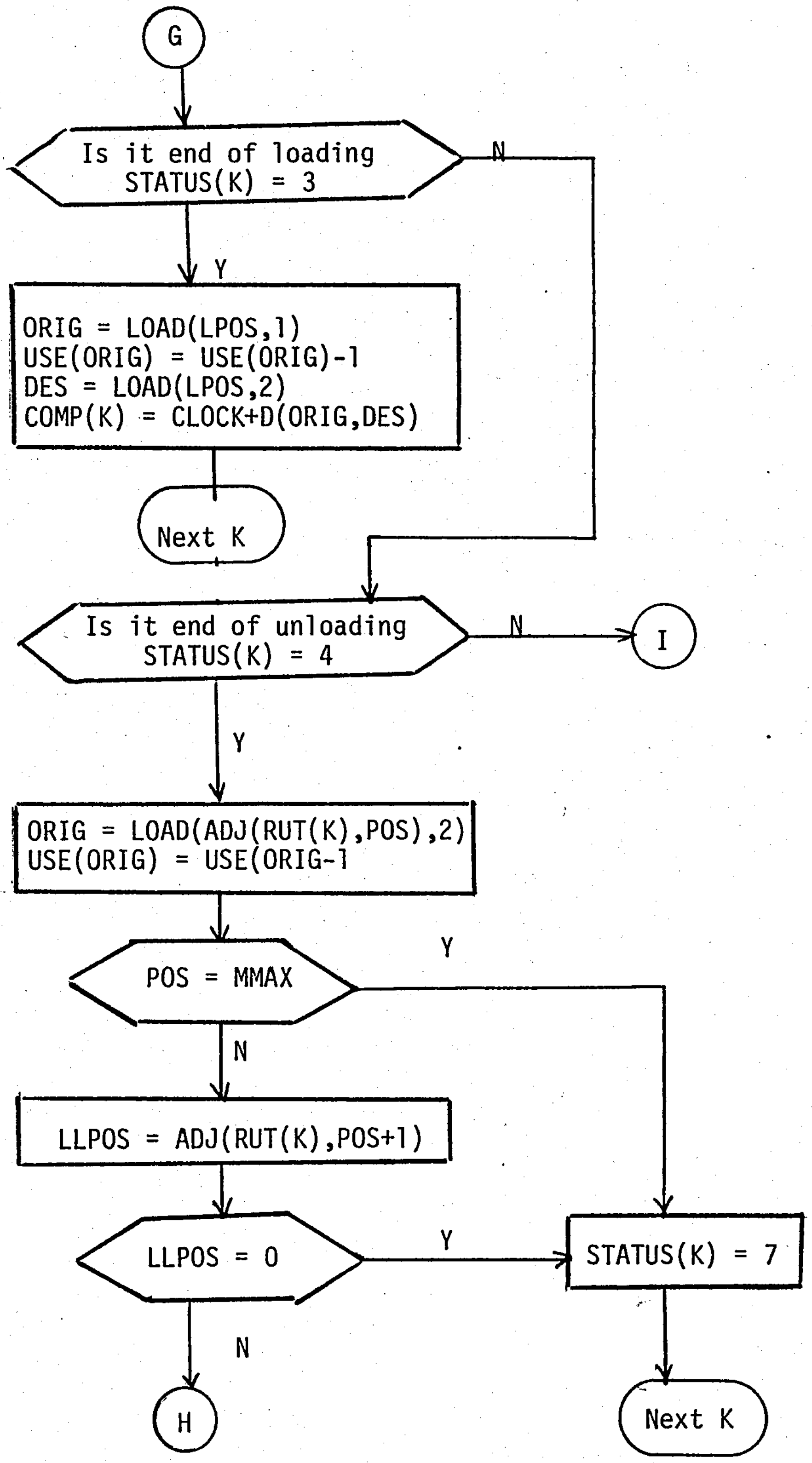


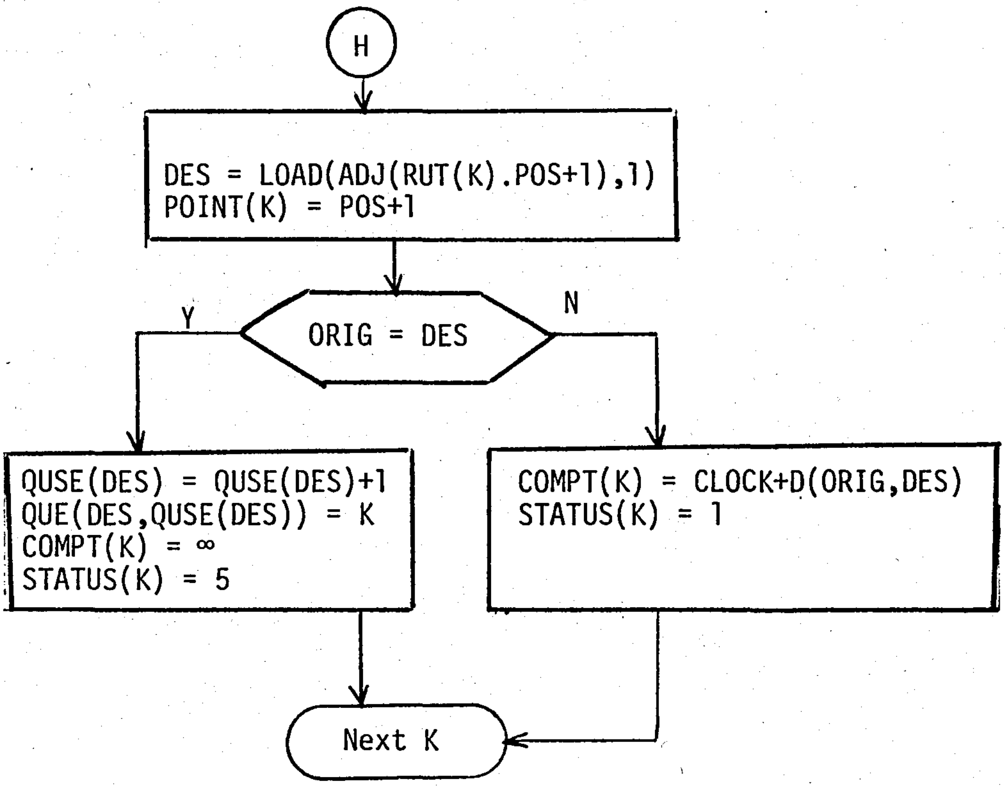
UPDATE SLACKS OF PLANES

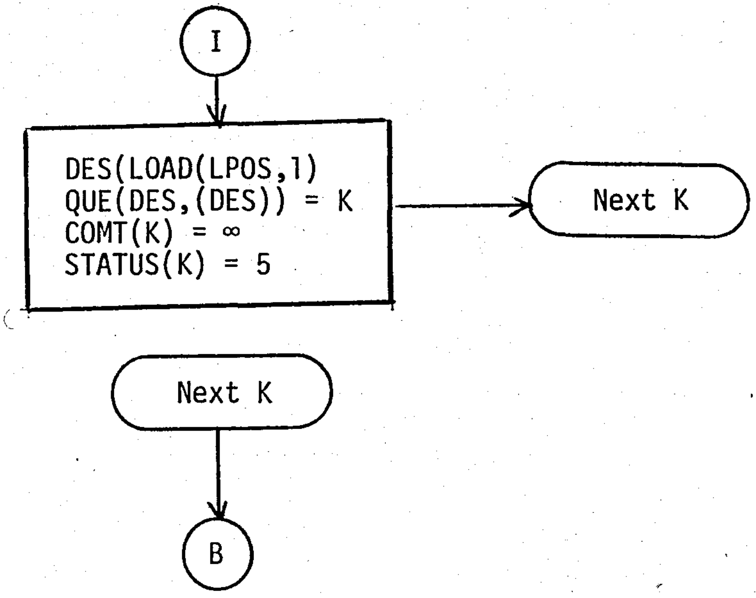


UPDATE PLANE STATUS

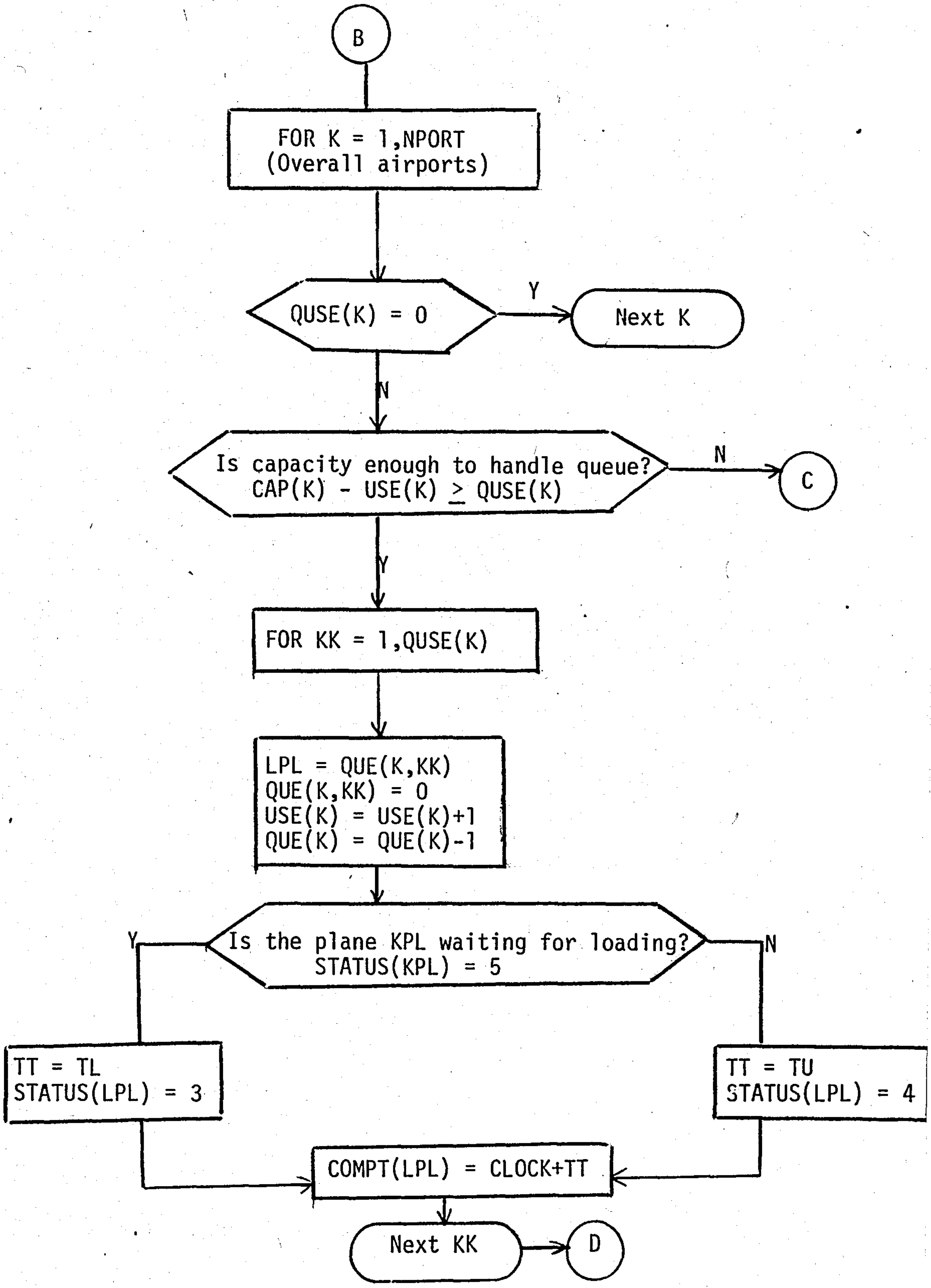


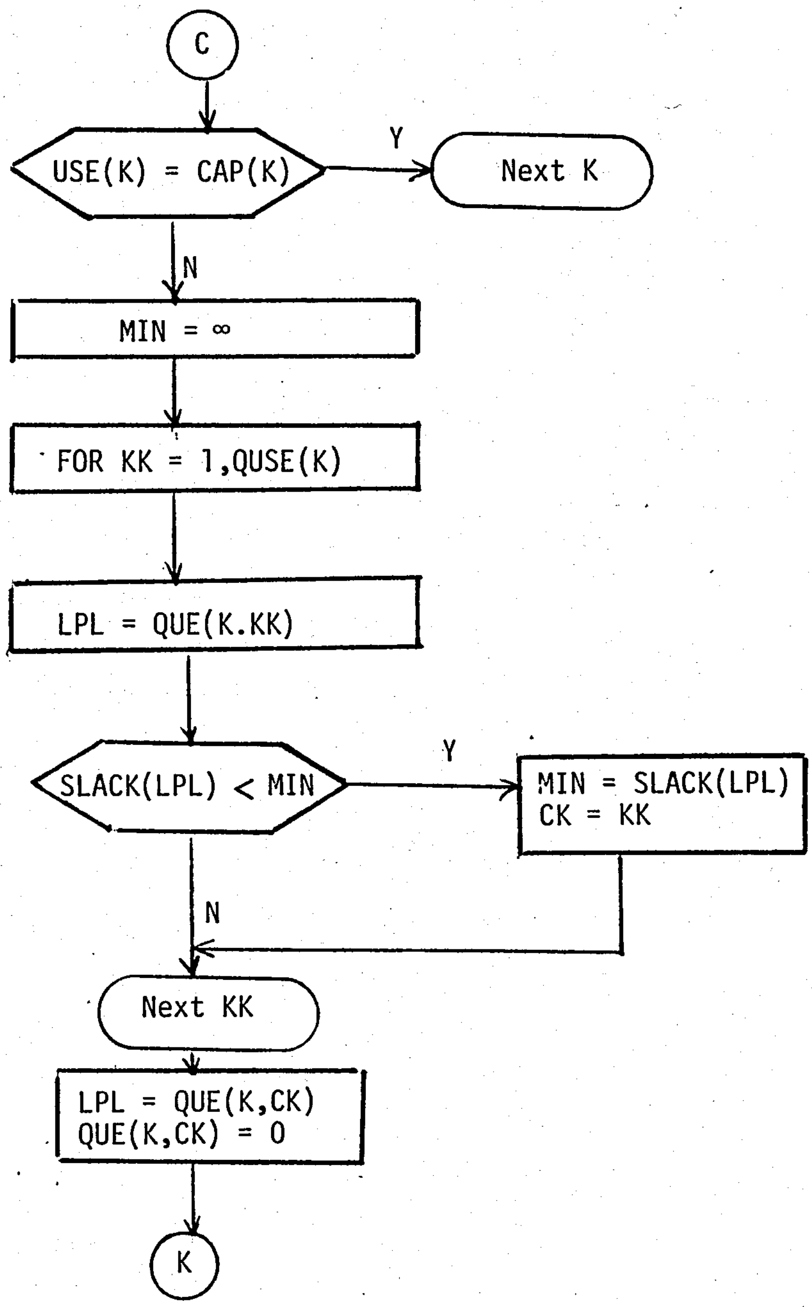

G
Is it end of loading STATUS(K) = 3
N
Y
ORIG = LOAD(LPOS,1)
USE(ORIG) = USE(ORIG)-1
DES = LOAD(LPOS,2)
COMP(K) = CLOCK+D(ORIG,DES)
Next K
Is it end of unloading STATUS(K) = 4
N
I
Y
ORIG = LOAD(ADJ(RUT(K),POS),2)
USE(ORIG) = USE(ORIG-1
POS = MMAX
Y
N
LLPOS = ADJ(RUT(K),POS+1)
LLPOS = 0
Y
STATUS(K) = 7
N
H
Next K

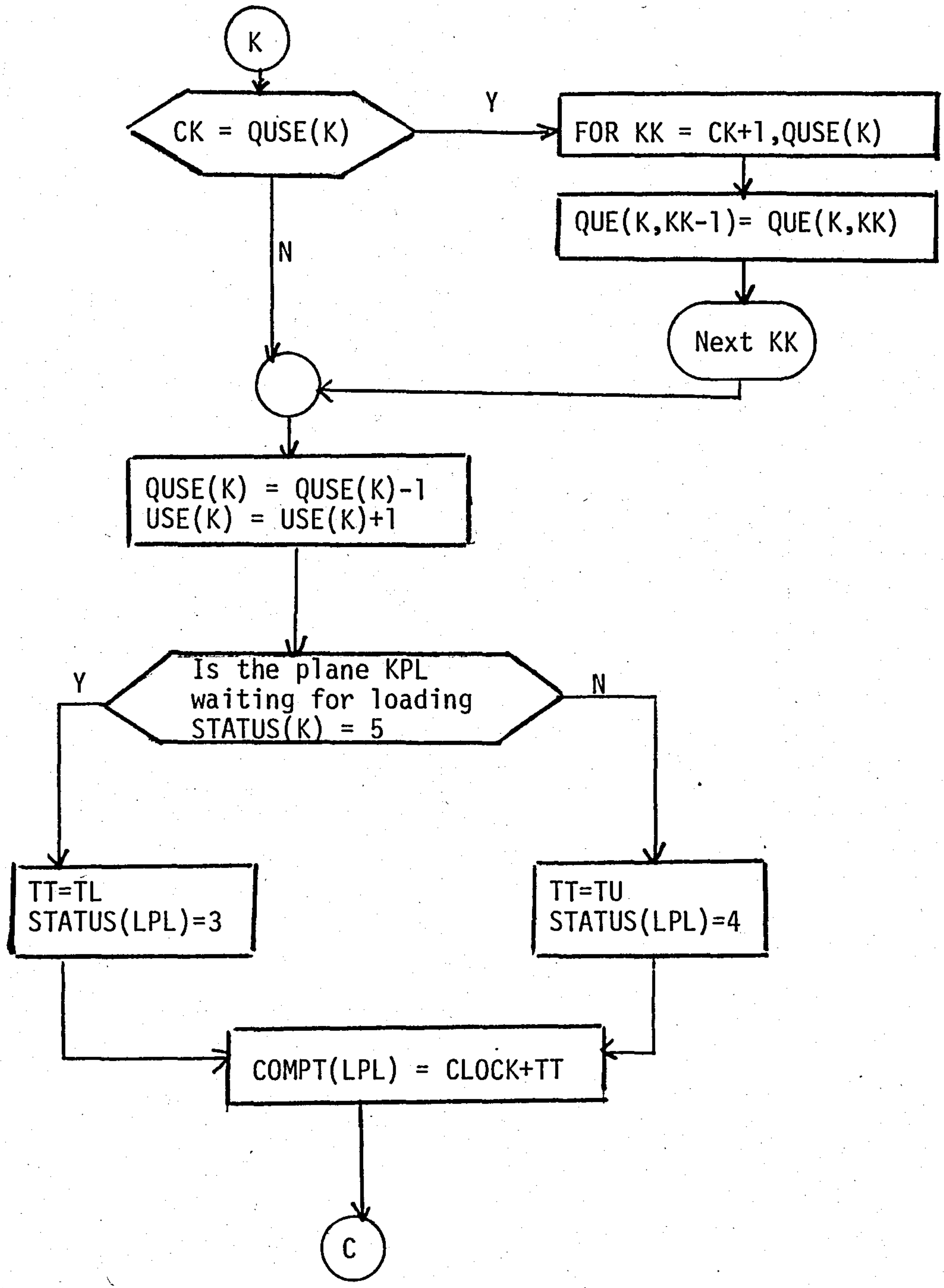

H
DES = LOAD(ADJ(RUT(K).POS+1),1)
POINT(K) = POS+1
ORIG = DES
Y
N
QUSE(DES) = QUSE(DES)+1
QUE(DES,QUSE(DES)) = K
COMPT(K) = ∞
STATUS(K) = 5
COMPT(K) = CLOCK+D(ORIG,DES)
STATUS(K) = 1
Next K

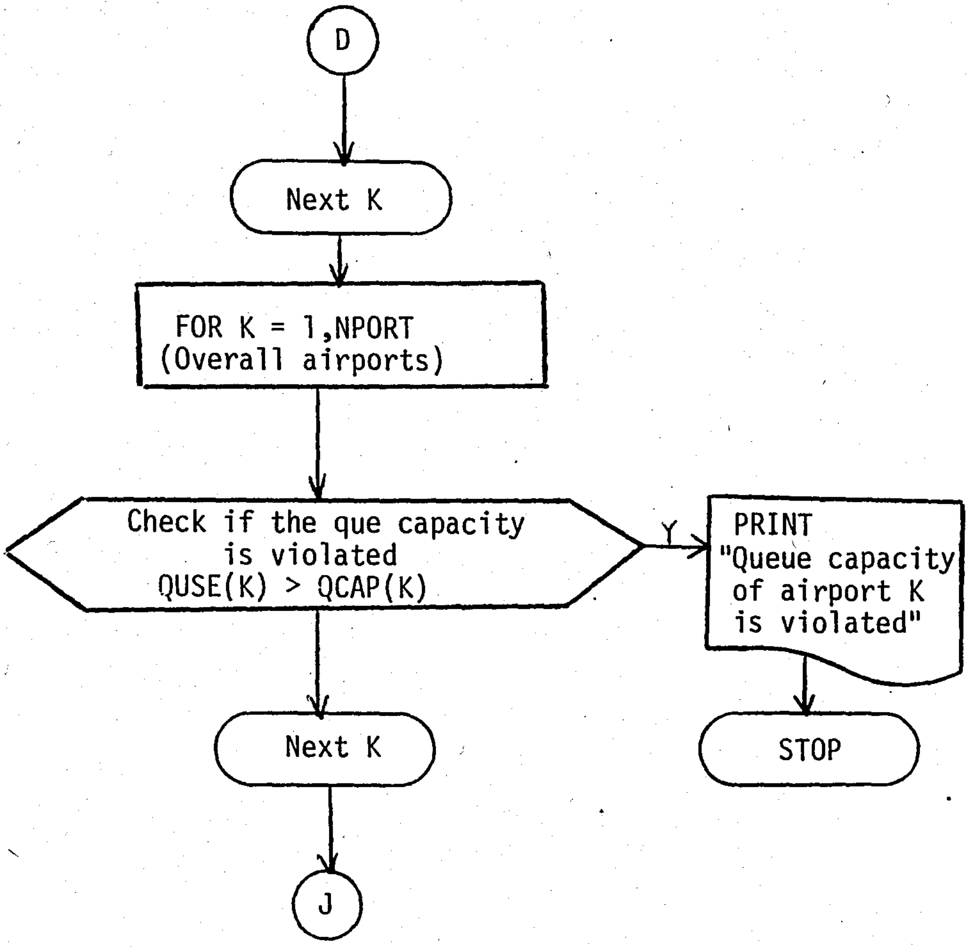

I
DES(LOAD(LPOS,1)
QUE(DES,(DES)) = K
COMT(K) = ∞
STATUS(K) = 5
Next K
Next K
B

DECIDE ON PRIORITY




C
USE(K) = CAP(K)
Y
Next K
N
MIN = ∞
FOR KK = 1,QUSE(K)
LPL = QUE(K.KK)
SLACK(LPL) < MIN
Y
MIN = SLACK(LPL)
CK = KK
N
Next KK
LPL = QUE(K,CK)
QUE(K,CK) = 0
K


K
CK = QUSE(K)
Y
N
FOR KK = CK+1,QUSE(K)
QUE(K,KK-1)= QUE(K,KK)
Next KK
QUSE(K) = QUSE(K)-1
USE(K) = USE(K)+1
Is the plane KPL waiting for loading STATUS(K) = 5
Y
N
TT=TL
STATUS(LPL)=3
TT=TU
STATUS(LPL)=4
COMPT(LPL) = CLOCK+TT
C


D
Next K
FOR K = 1,NPORT
(Overall airports)
Check if the que capacity
is violated
QUSE(K) > QCAP(K)
Y
PRINT
"Queue capacity
of airport K
is violated"
Next K
STOP
J

APPENDIX D

```
BOTTLENECK ROUTING OF CARGO AIRCRAFT
************************************


NUMBER OF PORTS        :   4
NUMBER OF LOADS        :  12
LOADING PLUS UNLOADING TIME :  15
MAXIMUM DISTANCE PERMITTED TO GENERATE PATHS : 150


AVAILABLE PLANES AT AIRPORTS
****************************
PORT   1            AVAILABLE PLANES    0
PORT   2            AVAILABLE PLANES    0
PORT   3            AVAILABLE PLANES    2
PORT   4            AVAILABLE PLANES    4
NUMBER OF PLANES :    6
TOTAL NUMBER OF NODES GENERATED :   16


MEANINGS OF NODES GENERATED
***************************

NODE    1 IS THE DUMMY SOURCE NODE
NODE    2 IS THE PORT    3
NODE    3 IS THE PORT    4
NODE    4 IS THE LOAD FROM PORT     1   TO PORT     2
NODE    5 IS THE LOAD FROM PORT     4   TO PORT     1
NODE    6 IS THE LOAD FROM PORT     2   TO PORT     4
NODE    7 IS THE LOAD FROM PORT     4   TO PORT     3
NODE    8 IS THE LOAD FROM PORT     4   TO PORT     2
NODE    9 IS THE LOAD FROM PORT     3   TO PORT     1
NODE   10 IS THE LOAD FROM PORT     3   TO PORT     4
NODE   11 IS THE LOAD FROM PORT     3   TO PORT     2
NODE   12 IS THE LOAD FROM PORT     2   TO PORT     1
NODE   13 IS THE LOAD FROM PORT     4   TO PORT     3
NODE   14 IS THE LOAD FROM PORT     1   TO PORT     3
NODE   15 IS THE LOAD FROM PORT     1   TO PORT     4
NODE   16 IS THE DUMMY TERMINAL NODE


LOADS BETWEEN PORTS
*******************

LOAD :   1    FROM PORT    1    TO PORT    2
LOAD :   2    FROM PORT    4    TO PORT    1
LOAD :   3    FROM PORT    2    TO PORT    4
LOAD :   4    FROM PORT    4    TO PORT    3
LOAD :   5    FROM PORT    4    TO PORT    2
LOAD :   6    FROM PORT    3    TO PORT    1
LOAD :   7    FROM PORT    3    TO PORT    4
LOAD :   8    FROM PORT    3    TO PORT    2
LOAD :   9    FROM PORT    2    TO PORT    1
LOAD :  10    FROM PORT    4    TO PORT    3
LOAD :  11    FROM PORT    1    TO PORT    3
LOAD :  12    FROM PORT    1    TO PORT    4
NUMBER OF PLANES :    6

NUMBER OF LOADS        :  12
```

```
        THE FEASIBILITY  CHECKS
        ***********************


        FROM HERE ON PATH LENGHTS ARE        35
        START CHECKING FOR FEASIBILITY

        NUMBER OF LOADS COVERED IS  0
        WHICH IS LESS THEN TOTAL LOADS 12
        RETURN FROM FEASIBILITY CHECK

  NO :  1, LBOY : 1, LEN :  35, INITIAL PORT : 3, LOADS COVERED   13
        FIRST OCURRANCE OF A  1 MEMBERED PATH
  NO :  2, LBOY : 1, LEN :  35, INITIAL PORT : 2, LOADS COVERED   10
  NO :  3, LBOY : 1, LEN :  35, INITIAL PORT : 3, LOADS COVERED    7
        FROM HERE ON PATH LENGHTS ARE        45
        START CHECKING FOR FEASIBILITY

        NUMBER OF LOADS COVERED IS  3
        WHICH IS LESS THEN TOTAL LOADS 12
        RETURN FROM FEASIBILITY CHECK

  NO :  4, LBOY : 1, LEN :  45, INITIAL PORT : 2, LOADS COVERED   11
        FROM HERE ON PATH LENGHTS ARE        50
        START CHECKING FOR FEASIBILITY

        NUMBER OF LOADS COVERED IS  4
        WHICH IS LESS THEN TOTAL LOADS 12
        RETURN FROM FEASIBILITY CHECK

  NO :  5, LBOY : 1, LEN :  50, INITIAL PORT : 3, LOADS COVERED    8
        FROM HERE ON PATH LENGHTS ARE        55
        START CHECKING FOR FEASIBILITY

        NUMBER OF LOADS COVERED IS  5
        WHICH IS LESS THEN TOTAL LOADS 12
        RETURN FROM FEASIBILITY CHECK

  NO :  6, LBOY : 1, LEN :  55, INITIAL PORT : 3, LOADS COVERED    5
  NO :  7, LBOY : 1, LEN :  55, INITIAL PORT : 2, LOADS COVERED    7
  NO :  8, LBOY : 1, LEN :  55, INITIAL PORT : 3, LOADS COVERED   10
  NO :  9, LBOY : 1, LEN :  55, INITIAL PORT : 2, LOADS COVERED   13
        FROM HERE ON PATH LENGHTS ARE        65
        START CHECKING FOR FEASIBILITY

        NUMBER OF LOADS COVERED IS  6
        WHICH IS LESS THEN TOTAL LOADS 12
        RETURN FROM FEASIBILITY CHECK

  NO : 10, LBOY : 1, LEN :  65, INITIAL PORT : 3, LOADS COVERED   1
        FROM HERE ON PATH LENGHTS ARE        70
        START CHECKING FOR FEASIBILITY

        NUMBER OF LOADS COVERED IS  6
        WHICH IS LESS THEN TOTAL LOADS 12
        RETURN FROM FEASIBILITY CHECK

  NO : 11, LBOY : 2, LEN :  70, INITIAL PORT : 3, LOADS COVERED   1
        FIRST OCURRANCE OF A  2 MEMBERED PATH
  NO : 12, LBOY : 2, LEN :  70, INITIAL PORT : 2, LOADS COVERED   1
  NO : 13, LBOY : 2, LEN :  70, INITIAL PORT : 3, LOADS COVERED
```

```
NO : 14, LBOY : 2, LEN :  70, INITIAL PORT : 2, LOADS COVERED   10    7
NO : 15, LBOY : 1, LEN :  70, INITIAL PORT : 2, LOADS COVERED    8
      FROM HERE ON PATH LENGHTS ARE        75
      START CHECKING FOR FEASIBILITY
      NUMBER OF LOADS COVERED IS  6
      WHICH IS LESS THEN TOTAL LOADS 12
      RETURN FROM FEASIBILITY CHECK

NO : 16, LBOY : 1, LEN :  75, INITIAL PORT : 2, LOADS COVERED    5
NO : 17, LBOY : 1, LEN :  75, INITIAL PORT : 2, LOADS COVERED   12
      FROM HERE ON PATH LENGHTS ARE        80
      START CHECKING FOR FEASIBILITY
      NUMBER OF LOADS COVERED IS  7
      WHICH IS LESS THEN TOTAL LOADS 12
      RETURN FROM FEASIBILITY CHECK

NO : 18, LBOY : 1, LEN :  80, INITIAL PORT : 3, LOADS COVERED   12
NO : 19, LBOY : 1, LEN :  80, INITIAL PORT : 2, LOADS COVERED    9
NO : 20, LBOY : 2, LEN :  80, INITIAL PORT : 3, LOADS COVERED    7   I1
NO : 21, LBOY : 2, LEN :  80, INITIAL PORT : 3, LOADS COVERED   13   I1
NO : 22, LBOY : 1, LEN :  80, INITIAL PORT : 2, LOADS COVERED    6
      FROM HERE ON PATH LENGHTS ARE        85
      START CHECKING FOR FEASIBILITY
      NUMBER OF LOADS COVERED IS  9
      WHICH IS LESS THEN TOTAL LOADS 12
      RETURN FROM FEASIBILITY CHECK

NO : 23, LBOY : 1, LEN :  85, INITIAL PORT : 3, LOADS COVERED    4
NO : 24, LBOY : 1, LEN :  85, INITIAL PORT : 3, LOADS COVERED    6
NO : 25, LBOY : 2, LEN :  85, INITIAL PORT : 2, LOADS COVERED   10    8
      FROM HERE ON PATH LENGHTS ARE        90
      START CHECKING FOR FEASIBILITY
      NUMBER OF LOADS COVERED IS 10
      WHICH IS LESS THEN TOTAL LOADS 12
      RETURN FROM FEASIBILITY CHECK

NO : 26, LBOY : 2, LEN :  90, INITIAL PORT : 2, LOADS COVERED   10    5
NO : 27, LBOY : 2, LEN :  90, INITIAL PORT : 2, LOADS COVERED   11   12
NO : 28, LBOY : 2, LEN :  90, INITIAL PORT : 3, LOADS COVERED    7   13
      FROM HERE ON PATH LENGHTS ARE        95
      START CHECKING FOR FEASIBILITY
      NUMBER OF LOADS COVERED IS 10
      WHICH IS LESS THEN TOTAL LOADS 12
      RETURN FROM FEASIBILITY CHECK

NO : 29, LBOY : 2, LEN :  95, INITIAL PORT : 2, LOADS COVERED   11    6
NO : 30, LBOY : 1, LEN :  95, INITIAL PORT : 3, LOADS COVERED   15
NO : 31, LBOY : 2, LEN :  95, INITIAL PORT : 3, LOADS COVERED    8   12
      FROM HERE ON PATH LENGHTS ARE       100
      START CHECKING FOR FEASIBILITY
      NUMBER OF LOADS COVERED IS 11
      WHICH IS LESS THEN TOTAL LOADS 12
```

```
       RETURN FROM FEASIBILITY CHECK

NO : 32, LBOY : 2, LEN : 100, INITIAL PORT : 3, LOADS COVERED    8   6
NO : 33, LBOY : 2, LEN : 100, INITIAL PORT : 2, LOADS COVERED    7  11
NO : 34, LBOY : 1, LEN : 100, INITIAL PORT : 3, LOADS COVERED    9
NO : 35, LBOY : 2, LEN : 100, INITIAL PORT : 3, LOADS COVERED    5   4
NO : 36, LBOY : 2, LEN : 100, INITIAL PORT : 2, LOADS COVERED   13  11
NO : 37, LBOY : 2, LEN : 100, INITIAL PORT : 2, LOADS COVERED   10  11
       FROM HERE ON PATH LENGHTS ARE      105
       START CHECKING FOR FEASIBILITY

       NUMBER OF LOADS COVERED IS 11
       WHICH IS LESS THEN TOTAL LOADS 12
       RETURN FROM FEASIBILITY CHECK

NO : 38, LBOY : 2, LEN : 105, INITIAL PORT : 3, LOADS COVERED    7   8
NO : 39, LBOY : 3, LEN : 105, INITIAL PORT : 3, LOADS COVERED    7  10
       FIRST OCURRANCE OF A  3 MEMBERED PATH
NO : 40, LBOY : 2, LEN : 105, INITIAL PORT : 3, LOADS COVERED   10   8
NO : 41, LBOY : 2, LEN : 105, INITIAL PORT : 3, LOADS COVERED   13   8
       FROM HERE ON PATH LENGHTS ARE      110
       START CHECKING FOR FEASIBILITY

       NUMBER OF LOADS COVERED IS 11
       WHICH IS LESS THEN TOTAL LOADS 12
       RETURN FROM FEASIBILITY CHECK

NO : 42, LBOY : 2, LEN : 110, INITIAL PORT : 3, LOADS COVERED    7  12
NO : 43, LBOY : 2, LEN : 110, INITIAL PORT : 3, LOADS COVERED    7   5
NO : 44, LBOY : 2, LEN : 110, INITIAL PORT : 3, LOADS COVERED    5  15
NO : 45, LBOY : 1, LEN : 110, INITIAL PORT : 2, LOADS COVERED    4
NO : 46, LBOY : 2, LEN : 110, INITIAL PORT : 3, LOADS COVERED   13   5
NO : 47, LBOY : 2, LEN : 110, INITIAL PORT : 3, LOADS COVERED   11  12
NO : 48, LBOY : 2, LEN : 110, INITIAL PORT : 3, LOADS COVERED   10   5
NO : 49, LBOY : 2, LEN : 110, INITIAL PORT : 2, LOADS COVERED    7  13
NO : 50, LBOY : 2, LEN : 110, INITIAL PORT : 3, LOADS COVERED   13  12
       FROM HERE ON PATH LENGHTS ARE      115
       START CHECKING FOR FEASIBILITY

       NUMBER OF LOADS COVERED IS 11
       WHICH IS LESS THEN TOTAL LOADS 12
       RETURN FROM FEASIBILITY CHECK

NO : 51, LBOY : 2, LEN : 115, INITIAL PORT : 3, LOADS COVERED    7   9
NO : 52, LBOY : 2, LEN : 115, INITIAL PORT : 2, LOADS COVERED   10  12
NO : 53, LBOY : 2, LEN : 115, INITIAL PORT : 3, LOADS COVERED   13   5
NO : 54, LBOY : 2, LEN : 115, INITIAL PORT : 3, LOADS COVERED   11   6
NO : 55, LBOY : 2, LEN : 115, INITIAL PORT : 3, LOADS COVERED    7   6
NO : 56, LBOY : 2, LEN : 115, INITIAL PORT : 2, LOADS COVERED    6   7
NO : 57, LBOY : 3, LEN : 115, INITIAL PORT : 2, LOADS COVERED   10  13
NO : 58, LBOY : 2, LEN : 115, INITIAL PORT : 3, LOADS COVERED   13   9
NO : 59, LBOY : 3, LEN : 115, INITIAL PORT : 2, LOADS COVERED   10   7  1
NO : 60, LBOY : 2, LEN : 115, INITIAL PORT : 2, LOADS COVERED    6  13
NO : 61, LBOY : 2, LEN : 115, INITIAL PORT : 2, LOADS COVERED    8  12
     FROM HERE ON PATH LENGHTS ARE      120
     START CHECKING FOR FEASIBILITY

     NUMBER OF LOADS COVERED IS 11
     WHICH IS LESS THEN TOTAL LOADS 12
     RETURN FROM FEASIBILITY CHECK

NO : 62, LBOY : 2, LEN : 120, INITIAL PORT : 2, LOADS COVERED   11   4
NO : 63, LBOY : 1, LEN : 120, INITIAL PORT : 3, LOADS COVERED   14


   ***THE FREQUENCY MATRIX***
 COL    1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
   ***************************************************
OW  1* 12 10 12  2  2  2  2  2  2  2  2  2  2  1  1  0
OW  2* 11 15 23  2  6  7 12  7  2 10 10  7 12  0  1  0
OW  3*  4  2  1  0  0  0  2  0  0  3  2  0  2  0  0  0
OW  4*  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
OW  5*  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0


   ***THE FIRST OCURRANCE MATRIX***
 COL    1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
   ***************************************************
OW  1*  1  2  1 23  6 22  3  5 19  2  4 17  1 53 30  0
OW  2* 11 12 11 35 26 29 13 25 51 11 20 27 11  0 44  0
OW  3* 39 57 39  0  0  0 39  0  0 39 57  0 39  0  0  0
OW  4*  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
OW  5*  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
NO : 64, LBOY : 3, LEN : 120, INITIAL PORT : 3, LOADS COVERED    7  10
NO : 65, LBOY : 2, LEN : 120, INITIAL PORT : 2, LOADS COVERED   10   6
NO : 66, LBOY : 2, LEN : 120, INITIAL PORT : 3, LOADS COVERED   10  11
NO : 67, LBOY : 2, LEN : 120, INITIAL PORT : 2, LOADS COVERED    8   6
NO : 68, LBOY : 3, LEN : 120, INITIAL PORT : 3, LOADS COVERED   13  10
NO : 69, LBOY : 1, LEN : 120, INITIAL PORT : 2, LOADS COVERED   15
NO : 70, LBOY : 2, LEN : 120, INITIAL PORT : 2, LOADS COVERED   10   4
NO : 71, LBOY : 2, LEN : 120, INITIAL PORT : 2, LOADS COVERED    5   4
NO : 72, LBOY : 2, LEN : 120, INITIAL PORT : 2, LOADS COVERED   12   4
     FROM HERE ON PATH LENGHTS ARE      125
     START CHECKING FOR FEASIBILITY

     AT LEAST  6  PLANE REQUIRED
     ALTHOUGH WE HAVE  6
```

```
          BLOCKING THE PATH LIST
          **********************


       ***THE FREQUENCY MATRIX***
    COL   1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
       ************************************************
 ROW  1* 12 11 12  2  2  2  2  2  2  2  2  2  2  1  2  0
 ROW  2* 11 20 24  5  7  9 12  8  2 13 11  8 12  0  1  0
 ROW  3*  5  2  3  0  0  0  3  2  0  5  2  0  3  0  0  0
 ROW  4*  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 ROW  5*  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0


       ***THE FIRST OCURRANCE MATRIX***
    COL   1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
       ************************************************
 ROW  1*  1  2  1 23  6 22  3  5 19  2  4 17  1 63 30  0
 ROW  2* 11 12 11 35 26 29 13 25 51 11 20 27 11  0 44  0
 ROW  3* 39 57 39  0  0  0 39 64  0 39 57  0 39  0  0  0
 ROW  4*  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 ROW  5*  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

          USET=  72

          TOTAL FREQUENCIES
          *****************

          LOAD : 1    TOTAL FREQUENCY :     7
          LOAD : 2    TOTAL FREQUENCY :     9
          LOAD : 3    TOTAL FREQUENCY :    11
          LOAD : 4    TOTAL FREQUENCY :    17
          LOAD : 5    TOTAL FREQUENCY :    12
          LOAD : 6    TOTAL FREQUENCY :     4
          LOAD : 7    TOTAL FREQUENCY :    20
          LOAD : 8    TOTAL FREQUENCY :    15
          LOAD : 9    TOTAL FREQUENCY :    10
          LOAD :10    TOTAL FREQUENCY :    17
          LOAD :11    TOTAL FREQUENCY :     1
          LOAD :12    TOTAL FREQUENCY :     3

          THE ADRES TABLE
          ***************

          NEW CODE :      1        OLD CODE :     2
          NEW CODE :      2        OLD CODE :     8
          NEW CODE :      3        OLD CODE :     1
          NEW CODE :      4        OLD CODE :     9
          NEW CODE :      5        OLD CODE :    11
          NEW CODE :      6        OLD CODE :    12
          NEW CODE :      7        OLD CODE :     3
          NEW CODE :      8        OLD CODE :     7
          NEW CODE :      9        OLD CODE :    13
          NEW CODE :     10        OLD CODE :    14
          NEW CODE :     11        OLD CODE :    28
          NEW CODE :     12        OLD CODE :    39
          NEW CODE :     13        OLD CODE :    49
          NEW CODE :     14        OLD CODE :     4
          NEW CODE :     15        OLD CODE :    10
          NEW CODE :     16        OLD CODE :    20
          NEW CODE :     17        OLD CODE :    21
          NEW CODE :     18        OLD CODE :    33
          NEW CODE :     19        OLD CODE :    36
          NEW CODE :     20        OLD CODE :    37
          NEW CODE :     21        OLD CODE :    57
          NEW CODE :     22        OLD CODE :    59
          NEW CODE :     23        OLD CODE :    66
          NEW CODE :     24        OLD CODE :     5
          NEW CODE :     25        OLD CODE :    15
          NEW CODE :     26        OLD CODE :    25
          NEW CODE :     27        OLD CODE :    38
          NEW CODE :     28        OLD CODE :    40
          NEW CODE :     29        OLD CODE :    41
          NEW CODE :     30        OLD CODE :    64
          NEW CODE :     31        OLD CODE :    68
          NEW CODE :     32        OLD CODE :    22
          NEW CODE :     33        OLD CODE :    24
          NEW CODE :     34        OLD CODE :    29
```

```
NEW CODE :  35     OLD CODE :  32
NEW CODE :  36     OLD CODE :  53
NEW CODE :  37     OLD CODE :  54
NEW CODE :  38     OLD CODE :  55
NEW CODE :  39     OLD CODE :  56
NEW CODE :  40     OLD CODE :  60
NEW CODE :  41     OLD CODE :  65
NEW CODE :  42     OLD CODE :  67
NEW CODE :  43     OLD CODE :  17
NEW CODE :  44     OLD CODE :  18
NEW CODE :  45     OLD CODE :  27
NEW CODE :  46     OLD CODE :  31
NEW CODE :  47     OLD CODE :  42
NEW CODE :  48     OLD CODE :  47
NEW CODE :  49     OLD CODE :  50
NEW CODE :  50     OLD CODE :  52
NEW CODE :  51     OLD CODE :  61
NEW CODE :  52     OLD CODE :   6
NEW CODE :  53     OLD CODE :  16
NEW CODE :  54     OLD CODE :  26
NEW CODE :  55     OLD CODE :  43
NEW CODE :  56     OLD CODE :  46
NEW CODE :  57     OLD CODE :  48
NEW CODE :  58     OLD CODE :  23
NEW CODE :  59     OLD CODE :  35
NEW CODE :  60     OLD CODE :  45
NEW CODE :  61     OLD CODE :  62
NEW CODE :  62     OLD CODE :  70
NEW CODE :  63     OLD CODE :  71
NEW CODE :  64     OLD CODE :  72
NEW CODE :  65     OLD CODE :  19
NEW CODE :  66     OLD CODE :  34
NEW CODE :  67     OLD CODE :  51
NEW CODE :  68     OLD CODE :  58
NEW CODE :  69     OLD CODE :  30
NEW CODE :  70     OLD CODE :  44
NEW CODE :  71     OLD CODE :  69
NEW CODE :  72     OLD CODE :  63
FIRST OCCJRRANCES OF MEMBER SIZES IN THE NEW PATH LIST
******************************************************

MEMBER SIZE : 1   FIRST OCCJRRANCE :    1
MEMBER SIZE : 2   FIRST OCCJRRANCE :    5
MEMBER SIZE : 3   FIRST OCCJRRANCE :   12

FIRST OCCJRRANCES OF LOADS IN THE NEW PATH LIST
***********************************************

LOAD : 1   FIRST OCCURRANCE :   58
LOAD : 2   FIRST OCCURRANCE :   52
LOAD : 3   FIRST OCCURRANCE :   32
LOAD : 4   FIRST OCCURRANCE :    7
LOAD : 5   FIRST OCCURRANCE :   24
LOAD : 6   FIRST OCCURRANCE :   65
LOAD : 7   FIRST OCCURRANCE :   [illegible]
LOAD : 8   FIRST OCCURRANCE :   14
LOAD : 9   FIRST OCCURRANCE :   43
LOAD :10   FIRST OCCURRANCE :    3
LOAD :11   FIRST OCCURRANCE :   72
LOAD :12   FIRST OCCURRANCE :   69
```

```
THE SEARCH STEP
***************

LIMITS FOR THE FIRST PLANE      KSET=  72        KRT=  72       YUK=   2
--------------------------------------------------------------------------
INCLUDE PATH :  63(  72)  PLANES USED= 1  LOADS COVERED=  1
PLANE ASSIGNMENTS:  72,
LOAD COVERAGE IS AS FOLLOWS :  0  0  0  0  0  0  0  0  0  0  1  0
THIS COVERAGE RESULTED WITH     KSET=  71        KRT=  69       YUK=   3
--------------------------------------------------------------------------
INCLUDE PATH :  69(  71)  PLANES USED= 2  LOADS COVERED=  2
PLANE ASSIGNMENTS:  72,  71,
LOAD COVERAGE IS AS FOLLOWS :  0  0  0  0  0  0  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH     KSET=  68        KRT=  65       YUK=   3
--------------------------------------------------------------------------
INCLUDE PATH :  58(  68)  PLANES USED= 3  LOADS COVERED=  4
PLANE ASSIGNMENTS:  72,  71,  68,
LOAD COVERAGE IS AS FOLLOWS :  0  0  0  0  0  1  0  0  0  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  54        KRT=  58       YUK=   3
--------------------------------------------------------------------------
INCLUDE PATH :  72(  64)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  68,  64,
LOAD COVERAGE IS AS FOLLOWS :  1  0  0  0  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  57        KRT=  52       YUK=   3
--------------------------------------------------------------------------
INCLUDE PATH :  48(  57)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  68,  64,  57,
LOAD COVERAGE IS AS FOLLOWS :  1  1  0  0  0  1  1  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  51        KRT=  72       YUK=   4
--------------------------------------------------------------------------
DELETE PATH :  48(  57)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  68,  64,
LOAD COVERAGE IS AS FOLLOWS :  1  0  0  0  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  56        KRT=  52       YUK=   3
--------------------------------------------------------------------------
INCLUDE PATH :  43(  55)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  68,  64,  55,
LOAD COVERAGE IS AS FOLLOWS :  1  1  0  1  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  51        KRT=  72       YUK=   4
--------------------------------------------------------------------------
DELETE PATH :  43(  55)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  68,  64,
LOAD COVERAGE IS AS FOLLOWS :  1  0  0  0  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  54        KRT=  52       YUK=   3
--------------------------------------------------------------------------
INCLUDE PATH :   6(  52)  PLANES USED= 5  LOADS COVERED=  7
PLANE ASSIGNMENTS:  72,  71,  68,  64,  52,
LOAD COVERAGE IS AS FOLLOWS :  1  1  0  0  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  51        KRT=  72       YUK=   5
--------------------------------------------------------------------------
DELETE PATH :   6(  52)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  68,  64,
LOAD COVERAGE IS AS FOLLOWS :  1  0  0  0  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  51        KRT=  52       YUK=   3
--------------------------------------------------------------------------
DELETE PATH :  72(  64)  PLANES USED= 3  LOADS COVERED=  4
PLANE ASSIGNMENTS:  72,  71,  68,
LOAD COVERAGE IS AS FOLLOWS :  0  0  0  0  0  1  0  0  0  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  63        KRT=  58       YUK=   3
--------------------------------------------------------------------------
INCLUDE PATH :  71(  63)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  68,  63,
LOAD COVERAGE IS AS FOLLOWS :  1  1  0  0  0  1  0  0  0  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  57        KRT=  43       YUK=   3
--------------------------------------------------------------------------
INCLUDE PATH :  47(  48)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  68,  63,  48,
LOAD COVERAGE IS AS FOLLOWS :  1  1  0  0  0  1  0  1  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  42        KRT=  72       YUK=   4
--------------------------------------------------------------------------
DELETE PATH :  47(  48)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  68,  63,
LOAD COVERAGE IS AS FOLLOWS :  1  1  0  0  0  1  0  0  0  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  47        KRT=  43       YUK=   3
--------------------------------------------------------------------------
INCLUDE PATH :  42(  47)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  68,  63,  47,
LOAD COVERAGE IS AS FOLLOWS :  1  1  0  1  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  42        KRT=  72       YUK=   4
--------------------------------------------------------------------------
DELETE PATH :  42(  47)  PLANES USED= 4  LOADS COVERED=  6
```

```
PLANE ASSIGNMENTS:  72,  71,  68,  63,
LOAD COVERAGE IS AS FOLLOWS :   1   1   0   0   0   1   0   0   0   1   1   1
THIS COVERAGE RESULTED WITH     KSET=  46        KRT=  43        YUK=   3
------------------------------------------------------------------------------
INCLUDE PATH :  31(  46)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  68,  63,  46,
LOAD COVERAGE IS AS FOLLOWS :   1   1   0   0   1   1   0   0   1   1   1   1
THIS COVERAGE RESULTED WITH     KSET=  42        KRT=  72        YUK=   4
------------------------------------------------------------------------------
DELETE PATH :  31(  46)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  68,  63,
LOAD COVERAGE IS AS FOLLOWS :   1   1   0   0   0   1   0   0   0   1   1   1
THIS COVERAGE RESULTED WITH     KSET=  45        KRT=  43        YUK=   3
------------------------------------------------------------------------------
INCLUDE PATH :  18(  44)  PLANES USED= 5  LOADS COVERED=  7
PLANE ASSIGNMENTS:  72,  71,  68,  63,  44,
LOAD COVERAGE IS AS FOLLOWS :   1   1   0   0   0   1   0   0   1   1   1   1
THIS COVERAGE RESULTED WITH     KSET=  42        KRT=  72        YUK=   5
------------------------------------------------------------------------------
DELETE PATH :  18(  44)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  68,  63,
LOAD COVERAGE IS AS FOLLOWS :   1   1   0   0   0   1   0   0   0   1   1   1
THIS COVERAGE RESULTED WITH     KSET=  43        KRT=  43        YUK=   3
------------------------------------------------------------------------------
DELETE PATH :  71(  63)  PLANES USED= 3  LOADS COVERED=  4
PLANE ASSIGNMENTS:  72,  71,  68,
LOAD COVERAGE IS AS FOLLOWS :   0   0   0   0   0   1   0   0   0   1   1   1
THIS COVERAGE RESULTED WITH     KSET=  62        KRT=  58        YUK=   3
------------------------------------------------------------------------------
INCLUDE PATH :  70(  62)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  68,  62,
LOAD COVERAGE IS AS FOLLOWS :   1   0   0   0   0   1   1   0   0   1   1   1
THIS COVERAGE RESULTED WITH     KSET=  57        KRT=  52        YUK=   3
------------------------------------------------------------------------------
INCLUDE PATH :  43(  55)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  68,  62,  55,
LOAD COVERAGE IS AS FOLLOWS :   1   1   0   1   0   1   1   0   0   1   1   1
THIS COVERAGE RESULTED WITH     KSET=  51        KRT=  72        YUK=   4
------------------------------------------------------------------------------
DELETE PATH :  43(  55)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  68,  62,
LOAD COVERAGE IS AS FOLLOWS :   1   0   0   0   0   1   1   0   0   1   1   1
THIS COVERAGE RESULTED WITH     KSET=  54        KRT=  52        YUK=   3
------------------------------------------------------------------------------
INCLUDE PATH :   6(  52)  PLANES USED= 5  LOADS COVERED=  7
PLANE ASSIGNMENTS:  72,  71,  68,  62,  52,
LOAD COVERAGE IS AS FOLLOWS :   1   1   0   0   0   1   1   0   0   1   1   1
THIS COVERAGE RESULTED WITH     KSET=  51        KRT=  72        YUK=   5
------------------------------------------------------------------------------
DELETE PATH :   6(  52)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  68,  62,
LOAD COVERAGE IS AS FOLLOWS :   1   0   0   0   0   1   1   0   0   1   1   1
THIS COVERAGE RESULTED WITH     KSET=  51        KRT=  52        YUK=   3
------------------------------------------------------------------------------
DELETE PATH :  70(  62)  PLANES USED= 3  LOADS COVERED=  4
PLANE ASSIGNMENTS:  72,  71,  68,
LOAD COVERAGE IS AS FOLLOWS :   0   0   0   0   0   1   0   0   0   1   1   1
THIS COVERAGE RESULTED WITH     KSET=  51        KRT=  58        YUK=   3
------------------------------------------------------------------------------
INCLUDE PATH :  62(  61)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  68,  61,
LOAD COVERAGE IS AS FOLLOWS :   1   0   0   0   0   1   0   1   0   1   1   1
THIS COVERAGE RESULTED WITH     KSET=  57        KRT=  52        YUK=   3
------------------------------------------------------------------------------
INCLUDE PATH :  48(  57)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  68,  61,  57,
LOAD COVERAGE IS AS FOLLOWS :   1   1   0   0   0   1   1   1   0   1   1   1
THIS COVERAGE RESULTED WITH     KSET=  51        KRT=  72        YUK=   4
------------------------------------------------------------------------------
DELETE PATH :  48(  57)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  68,  61,
LOAD COVERAGE IS AS FOLLOWS :   1   0   0   0   0   1   0   1   0   1   1   1
THIS COVERAGE RESULTED WITH     KSET=  56        KRT=  52        YUK=   3
------------------------------------------------------------------------------
INCLUDE PATH :  43(  55)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  68,  61,  55,
LOAD COVERAGE IS AS FOLLOWS :   1   1   0   1   0   1   0   1   0   1   1   1
THIS COVERAGE RESULTED WITH     KSET=  51        KRT=  72        YUK=   4
------------------------------------------------------------------------------
DELETE PATH :  43(  55)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  68,  61,
LOAD COVERAGE IS AS FOLLOWS :   1   0   0   0   0   1   0   1   0   1   1   1
```

```
THIS COVERAGE RESULTED WITH    KSET=  54        KRT=  52        YUK=   3
-------------------------------------------------------------------------
INCLUDE PATH :    6(  52)  PLANES USED= 5  LOADS COVERED=  7
PLANE ASSIGNMENTS:  72,  71,  68,  61,  52,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  0  0  1  0  1  0  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  51        KRT=  72        YUK=   5
-------------------------------------------------------------------------
DELETE PATH :    6(  52)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  68,  61,
LOAD COVERAGE IS AS FOLLOWS :   1  0  0  0  0  1  0  1  0  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  51        KRT=  52        YUK=   3
-------------------------------------------------------------------------
DELETE PATH :   52(  61)  PLANES USED= 3  LOADS COVERED=  4
PLANE ASSIGNMENTS:  72,  71,  68,
LOAD COVERAGE IS AS FOLLOWS :   0  0  0  0  0  1  0  0  0  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  60        KRT=  58        YUK=   3
-------------------------------------------------------------------------
INCLUDE PATH :   45(  60)  PLANES USED= 4  LOADS COVERED=  5
PLANE ASSIGNMENTS:  72,  71,  68,  60,
LOAD COVERAGE IS AS FOLLOWS :   1  0  0  0  0  1  0  0  0  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  57        KRT=  72        YUK=   4
-------------------------------------------------------------------------
DELETE PATH :   45(  60)  PLANES USED= 3  LOADS COVERED=  4
PLANE ASSIGNMENTS:  72,  71,  68,
LOAD COVERAGE IS AS FOLLOWS :   0  0  0  0  0  1  0  0  0  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  59        KRT=  58        YUK=   3
-------------------------------------------------------------------------
INCLUDE PATH :   35(  59)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  68,  59,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  0  0  1  0  0  0  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  57        KRT=  43        YUK=   3
-------------------------------------------------------------------------
INCLUDE PATH :   61(  51)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  68,  59,  51,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  0  1  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  42        KRT=  72        YUK=   4
-------------------------------------------------------------------------
DELETE PATH :   61(  51)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  68,  59,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  0  0  1  0  0  0  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  50        KRT=  43        YUK=   3
-------------------------------------------------------------------------
INCLUDE PATH :   52(  50)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  68,  59,  50,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  0  0  1  1  0  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  42        KRT=  72        YUK=   4
-------------------------------------------------------------------------
DELETE PATH :   52(  50)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  68,  59,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  0  0  1  0  0  0  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  49        KRT=  43        YUK=   3
-------------------------------------------------------------------------
INCLUDE PATH :   47(  48)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  68,  59,  48,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  0  0  1  0  1  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  42        KRT=  72        YUK=   4
-------------------------------------------------------------------------
DELETE PATH :   47(  48)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  68,  59,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  0  0  1  0  0  0  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  47        KRT=  43        YUK=   3
-------------------------------------------------------------------------
INCLUDE PATH :   42(  47)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  68,  59,  47,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  1  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  42        KRT=  72        YUK=   4
-------------------------------------------------------------------------
DELETE PATH :   42(  47)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  68,  59,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  0  0  1  0  0  0  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  46        KRT=  43        YUK=   3
-------------------------------------------------------------------------
INCLUDE PATH :   31(  46)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  68,  59,  46,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  0  1  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  42        KRT=  72        YUK=   4
-------------------------------------------------------------------------
DELETE PATH :   31(  46)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  68,  59,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  0  0  1  0  0  0  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  45        KRT=  43        YUK=   3
-------------------------------------------------------------------------
```

```
INCLUDE PATH :  27(  45)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  68,  59,  45,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  0  0  1  0  1  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  42      KRT=  72      YUK=   4
---------------------------------------------------------------------
DELETE PATH :  27(  45)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  68,  59,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  0  0  1  0  0  0  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  44      KRT=  43      YUK=   3
---------------------------------------------------------------------
INCLUDE PATH :  18(  44)  PLANES USED= 5  LOADS COVERED=  7
PLANE ASSIGNMENTS:  72,  71,  68,  59,  44,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  0  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  42      KRT=  72      YUK=   5
---------------------------------------------------------------------
DELETE PATH :  18(  44)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  68,  59,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  0  0  1  0  0  0  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  43      KRT=  43      YUK=   3
---------------------------------------------------------------------
INCLUDE PATH :  17(  43)  PLANES USED= 5  LOADS COVERED=  7
PLANE ASSIGNMENTS:  72,  71,  68,  59,  43,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  0  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  42      KRT=  72      YUK=   5
---------------------------------------------------------------------
DELETE PATH :  17(  43)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  68,  59,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  0  0  1  0  0  0  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  42      KRT=  43      YUK=   3
---------------------------------------------------------------------
DELETE PATH :  35(  59)  PLANES USED= 3  LOADS COVERED=  4
PLANE ASSIGNMENTS:  72,  71,  68,
LOAD COVERAGE IS AS FOLLOWS :   0  0  0  0  0  1  0  0  0  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  58      KRT=  58      YUK=   3
---------------------------------------------------------------------
INCLUDE PATH :  23(  58)  PLANES USED= 4  LOADS COVERED=  5
PLANE ASSIGNMENTS:  72,  71,  68,  58,
LOAD COVERAGE IS AS FOLLOWS :   1  0  0  0  0  1  0  0  0  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  57      KRT=  72      YUK=   4
---------------------------------------------------------------------
DELETE PATH :  23(  58)  PLANES USED= 3  LOADS COVERED=  4
PLANE ASSIGNMENTS:  72,  71,  68,
LOAD COVERAGE IS AS FOLLOWS :   0  0  0  0  0  1  0  0  0  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  57      KRT=  58      YUK=   3
---------------------------------------------------------------------
DELETE PATH :  58(  68)  PLANES USED= 2  LOADS COVERED=  2
PLANE ASSIGNMENTS:  72,  71,
LOAD COVERAGE IS AS FOLLOWS :   0  0  0  0  0  0  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  57      KRT=  65      YUK=   3
---------------------------------------------------------------------
INCLUDE PATH :  51(  67)  PLANES USED= 3  LOADS COVERED=  4
PLANE ASSIGNMENTS:  72,  71,  67,
LOAD COVERAGE IS AS FOLLOWS :   0  0  0  1  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  64      KRT=  58      YUK=   3
---------------------------------------------------------------------
INCLUDE PATH :  72(  64)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  67,  64,
LOAD COVERAGE IS AS FOLLOWS :   1  0  0  1  0  1  0  0  1  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  57      KRT=  52      YUK=   3
---------------------------------------------------------------------
INCLUDE PATH :  48(  57)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  67,  64,  57,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  1  0  1  1  0  1  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  51      KRT=  72      YUK=   4
---------------------------------------------------------------------
DELETE PATH :  48(  57)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  67,  64,
LOAD COVERAGE IS AS FOLLOWS :   1  0  0  1  0  1  0  0  1  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  56      KRT=  52      YUK=   3
---------------------------------------------------------------------
INCLUDE PATH :  46(  56)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  67,  64,  56,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  1  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  51      KRT=  72      YUK=   4
---------------------------------------------------------------------
DELETE PATH :  46(  56)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  67,  64,
LOAD COVERAGE IS AS FOLLOWS :   1  0  0  1  0  1  0  0  1  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  55      KRT=  52      YUK=   3
---------------------------------------------------------------------
INCLUDE PATH :   6(  52)  PLANES USED= 5  LOADS COVERED=  7
PLANE ASSIGNMENTS:  72,  71,  67,  64,  52,
LOAD COVERAGE IS AS FOLLOWS :   1  1   0  1  0  1  0  0  1  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  51        KRT=  72        YUK=   5
-----------------------------------------------------------------------
DELETE PATH :   6(  52)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  67,  64,
LOAD COVERAGE IS AS FOLLOWS :   1  0   0  1  0  1  0  0  1  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  51        KRT=  52        YUK=   3
-----------------------------------------------------------------------
DELETE PATH :  72(  64)  PLANES USED= 3  LOADS COVERED=  4
PLANE ASSIGNMENTS:  72,  71,  67,
LOAD COVERAGE IS AS FOLLOWS :   0  0   0  1  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  63        KRT=  58        YUK=   3
-----------------------------------------------------------------------
INCLUDE PATH :  71(  63)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  67,  63,
LOAD COVERAGE IS AS FOLLOWS :   1  1   0  1  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  57        KRT=  43        YUK=   3
-----------------------------------------------------------------------
INCLUDE PATH :  50(  49)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  67,  63,  49,
LOAD COVERAGE IS AS FOLLOWS :   1  1   0  1  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  42        KRT=  72        YUK=   4
-----------------------------------------------------------------------
DELETE PATH :  50(  49)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  67,  63,
LOAD COVERAGE IS AS FOLLOWS :   1  1   0  1  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  48        KRT=  43        YUK=   3
-----------------------------------------------------------------------
INCLUDE PATH :  47(  48)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  67,  63,  48,
LOAD COVERAGE IS AS FOLLOWS :   1  1   0  1  0  1  0  1  1  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  42        KRT=  72        YUK=   4
-----------------------------------------------------------------------
DELETE PATH :  47(  48)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  67,  63,
LOAD COVERAGE IS AS FOLLOWS :   1  1   0  1  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  47        KRT=  43        YUK=   3
-----------------------------------------------------------------------
INCLUDE PATH :  31(  46)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  67,  63,  46,
LOAD COVERAGE IS AS FOLLOWS :   1  1   0  1  1  1  0  0  1  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  42        KRT=  72        YUK=   4
-----------------------------------------------------------------------
DELETE PATH :  31(  46)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  67,  63,
LOAD COVERAGE IS AS FOLLOWS :   1  1   0  1  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  45        KRT=  43        YUK=   3
-----------------------------------------------------------------------
INCLUDE PATH :  18(  44)  PLANES USED= 5  LOADS COVERED=  7
PLANE ASSIGNMENTS:  72,  71,  67,  63,  44,
LOAD COVERAGE IS AS FOLLOWS :   1  1   0  1  0  1  0  0  1  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  42        KRT=  72        YUK=   5
-----------------------------------------------------------------------
DELETE PATH :  18(  44)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  67,  63,
LOAD COVERAGE IS AS FOLLOWS :   1  1   0  1  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  43        KRT=  43        YUK=   3
-----------------------------------------------------------------------
DELETE PATH :  71(  63)  PLANES USED= 3  LOADS COVERED=  4
PLANE ASSIGNMENTS:  72,  71,  67,
LOAD COVERAGE IS AS FOLLOWS :   0  0   0  1  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  62        KRT=  58        YUK=   3
-----------------------------------------------------------------------
INCLUDE PATH :  70(  62)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  67,  62,
LOAD COVERAGE IS AS FOLLOWS :   1  0   0  1  0  1  1  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  57        KRT=  52        YUK=   3
-----------------------------------------------------------------------
INCLUDE PATH :  46(  56)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  67,  62,  56,
LOAD COVERAGE IS AS FOLLOWS :   1  1   0  1  0  1  1  0  0  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  51        KRT=  72        YUK=   4
-----------------------------------------------------------------------
DELETE PATH :  46(  56)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  67,  62,
LOAD COVERAGE IS AS FOLLOWS :   1  0   0  1  0  1  1  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  55        KRT=  52        YUK=   3
-----------------------------------------------------------------------
INCLUDE PATH :   6(  52)  PLANES USED= 5  LOADS COVERED=  7
PLANE ASSIGNMENTS:  72,  71,  67,  62,  52,
LOAD COVERAGE IS AS FOLLOWS :   1  1   0  1  0  1  1  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  51        KRT=  72        YUK=   5
```

```
-------------------------------------------------------------------------------
DELETE PATH :   6(  52)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  67,  62,
LOAD COVERAGE IS AS FOLLOWS :  1  0  0  1  0  1  1  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  51        KRT=  52       YUK=   3
-------------------------------------------------------------------------------
DELETE PATH :  70(  62)  PLANES USED= 3  LOADS COVERED=  4
PLANE ASSIGNMENTS:  72,  71,  67,
LOAD COVERAGE IS AS FOLLOWS :  0  0  0  1  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  51        KRT=  58       YUK=   3
-------------------------------------------------------------------------------
INCLUDE PATH :  62(  61)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  67,  61,
LOAD COVERAGE IS AS FOLLOWS :  1  0  0  1  0  1  0  1  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  57        KRT=  52       YUK=   3
-------------------------------------------------------------------------------
INCLUDE PATH :  48(  57)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  67,  61,  57,
LOAD COVERAGE IS AS FOLLOWS :  1  1  0  1  0  1  1  1  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  51        KRT=  72       YUK=   4
-------------------------------------------------------------------------------
DELETE PATH :  48(  57)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  67,  61,
LOAD COVERAGE IS AS FOLLOWS :  1  0  0  1  0  1  0  1  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  56        KRT=  52       YUK=   3
-------------------------------------------------------------------------------
INCLUDE PATH :  46(  56)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  67,  61,  56,
LOAD COVERAGE IS AS FOLLOWS :  1  1  0  1  0  1  0  1  0  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  51        KRT=  72       YUK=   4
-------------------------------------------------------------------------------
DELETE PATH :  46(  56)  PLANES USED= 4  LOADS COVERED=  5
PLANE ASSIGNMENTS:  72,  71,  67,  61,
LOAD COVERAGE IS AS FOLLOWS :  1  0  0  1  0  1  0  1  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  55        KRT=  52       YUK=   3
-------------------------------------------------------------------------------
INCLUDE PATH :   6(  52)  PLANES USED= 5  LOADS COVERED=  7
PLANE ASSIGNMENTS:  72,  71,  67,  61,  52,
LOAD COVERAGE IS AS FOLLOWS :  1  1  0  1  0  1  0  1  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  51        KRT=  72       YUK=   5
-------------------------------------------------------------------------------
DELETE PATH :   6(  52)  PLANES USED= 4  LOADS COVERED=  5
PLANE ASSIGNMENTS:  72,  71,  67,  61,
LOAD COVERAGE IS AS FOLLOWS :  1  0  0  1  0  1  0  1  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  51        KRT=  52       YUK=   3
-------------------------------------------------------------------------------
DELETE PATH :  62(  61)  PLANES USED= 3  LOADS COVERED=  4
PLANE ASSIGNMENTS:  72,  71,  67,
LOAD COVERAGE IS AS FOLLOWS :  0  0  0  1  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  60        KRT=  58       YUK=   3
-------------------------------------------------------------------------------
INCLUDE PATH :  45(  60)  PLANES USED= 4  LOADS COVERED=  5
PLANE ASSIGNMENTS:  72,  71,  67,  60,
LOAD COVERAGE IS AS FOLLOWS :  1  0  0  1  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  57        KRT=  72       YUK=   4
-------------------------------------------------------------------------------
DELETE PATH :  45(  60)  PLANES USED= 3  LOADS COVERED=  4
PLANE ASSIGNMENTS:  72,  71,  67,
LOAD COVERAGE IS AS FOLLOWS :  0  0  0  1  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  59        KRT=  58       YUK=   3
-------------------------------------------------------------------------------
INCLUDE PATH :  35(  59)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  67,  59,
LOAD COVERAGE IS AS FOLLOWS :  1  1  0  1  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  57        KRT=  43       YUK=   3
-------------------------------------------------------------------------------
INCLUDE PATH :  61(  51)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  67,  59,  51,
LOAD COVERAGE IS AS FOLLOWS :  1  1  0  1  1  1  0  0  1  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  42        KRT=  72       YUK=   4
-------------------------------------------------------------------------------
DELETE PATH :  61(  51)  PLANES USED= 4  LOADS COVERED=  5
PLANE ASSIGNMENTS:  72,  71,  67,  59,
LOAD COVERAGE IS AS FOLLOWS :  1  1  0  1  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  50        KRT=  43       YUK=   3
-------------------------------------------------------------------------------
INCLUDE PATH :  52(  50)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  67,  59,  50,
LOAD COVERAGE IS AS FOLLOWS :  1  1  0  1  0  1  1  0  1  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  42        KRT=  72       YUK=   4
-------------------------------------------------------------------------------
DELETE PATH :  52(  50)  PLANES USED= 4  LOADS COVERED=  6
```

```
PLANE ASSIGNMENTS:  72,  71,  67,  59,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  1  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  49        KRT=  43       YUK=   3
-------------------------------------------------------------------------
INCLUDE PATH :  50(  49)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  67,  59,  49,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  1  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  42        KRT=  72       YUK=   4
-------------------------------------------------------------------------
DELETE PATH :  50(  49)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  67,  59,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  1  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  48        KRT=  43       YUK=   3
-------------------------------------------------------------------------
INCLUDE PATH :  47(  48)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  67,  59,  48,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  1  0  1  0  1  1  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  42        KRT=  72       YUK=   4
-------------------------------------------------------------------------
DELETE PATH :  47(  48)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  67,  59,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  1  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  47        KRT=  43       YUK=   3
-------------------------------------------------------------------------
INCLUDE PATH :  31(  46)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  67,  59,  46,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  1  1  1  0  0  1  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  42        KRT=  72       YUK=   4
-------------------------------------------------------------------------
DELETE PATH :  31(  46)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  67,  59,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  1  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  45        KRT=  43       YUK=   3
-------------------------------------------------------------------------
INCLUDE PATH :  27(  45)  PLANES USED= 5  LOADS COVERED=  8
PLANE ASSIGNMENTS:  72,  71,  67,  59,  45,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  1  0  1  0  1  1  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  42        KRT=  72       YUK=   4
-------------------------------------------------------------------------
DELETE PATH :  27(  45)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  67,  59,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  1  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  44        KRT=  43       YUK=   3
-------------------------------------------------------------------------
INCLUDE PATH :  18(  44)  PLANES USED= 5  LOADS COVERED=  7
PLANE ASSIGNMENTS:  72,  71,  67,  59,  44,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  1  0  1  0  0  1  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  42        KRT=  72       YUK=   5
-------------------------------------------------------------------------
DELETE PATH :  18(  44)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  67,  59,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  1  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  43        KRT=  43       YUK=   3
-------------------------------------------------------------------------
INCLUDE PATH :  17(  43)  PLANES USED= 5  LOADS COVERED=  7
PLANE ASSIGNMENTS:  72,  71,  67,  59,  43,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  1  0  1  0  0  1  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  42        KRT=  72       YUK=   5
-------------------------------------------------------------------------
DELETE PATH :  17(  43)  PLANES USED= 4  LOADS COVERED=  6
PLANE ASSIGNMENTS:  72,  71,  67,  59,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  1  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  42        KRT=  43       YUK=   3
-------------------------------------------------------------------------
DELETE PATH :  35(  59)  PLANES USED= 3  LOADS COVERED=  4
PLANE ASSIGNMENTS:  72,  71,  67,
LOAD COVERAGE IS AS FOLLOWS :   0  0  0  1  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  58        KRT=  58       YUK=   3
-------------------------------------------------------------------------
INCLUDE PATH :  23(  58)  PLANES USED= 4  LOADS COVERED=  5
PLANE ASSIGNMENTS:  72,  71,  67,  58,
LOAD COVERAGE IS AS FOLLOWS :   1  0  0  1  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  57        KRT=  72       YUK=   4
-------------------------------------------------------------------------
DELETE PATH :  23(  58)  PLANES USED= 3  LOADS COVERED=  4
PLANE ASSIGNMENTS:  72,  71,  67,
LOAD COVERAGE IS AS FOLLOWS :   0  0  0  1  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  57        KRT=  58       YUK=   3
-------------------------------------------------------------------------
DELETE PATH :  51(  67)  PLANES USED= 2  LOADS COVERED=  2
PLANE ASSIGNMENTS:  72,  71,
LOAD COVERAGE IS AS FOLLOWS :   0  0  0  0  0  0  0  0  0  0  1  1
```

```
THIS COVERAGE RESULTED WITH    KSET=  66          KRT=  65        YUK=    3
------------------------------------------------------------------------------
INCLUDE PATH :  34(  66)  PLANES USED= 3  LOADS COVERED=  3
PLANE ASSIGNMENTS:  72,  71,  66,
LOAD COVERAGE IS AS FOLLOWS :   0  0   0  0  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH     KSET=  64          KRT=  58        YUK=    3
------------------------------------------------------------------------------
INCLUDE PATH :  72(  64)  PLANES USED= 4  LOADS COVERED=  5
PLANE ASSIGNMENTS:  72,  71,  66,  64,
LOAD COVERAGE IS AS FOLLOWS :   1  0   0  0  0  1  0  0  1  0  1  1
THIS COVERAGE RESULTED WITH     KSET=  57          KRT=  72        YUK=    4
------------------------------------------------------------------------------
DELETE PATH :  72(  64)  PLANES USED= 3  LOADS COVERED=  3
PLANE ASSIGNMENTS:  72,  71,  66,
LOAD COVERAGE IS AS FOLLOWS :   0  0   0  0  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH     KSET=  63          KRT=  58        YUK=    3
------------------------------------------------------------------------------
INCLUDE PATH :  71(  63)  PLANES USED= 4  LOADS COVERED=  5
PLANE ASSIGNMENTS:  72,  71,  66,  63,
LOAD COVERAGE IS AS FOLLOWS :   1  1   0  0  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH     KSET=  57          KRT=  72        YUK=    4
------------------------------------------------------------------------------
DELETE PATH :  71(  63)  PLANES USED= 3  LOADS COVERED=  3
PLANE ASSIGNMENTS:  72,  71,  66,
LOAD COVERAGE IS AS FOLLOWS :   0  0   0  0  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH     KSET=  62          KRT=  58        YUK=    3
------------------------------------------------------------------------------
INCLUDE PATH :  70(  62)  PLANES USED= 4  LOADS COVERED=  5
PLANE ASSIGNMENTS:  72,  71,  66,  62,
LOAD COVERAGE IS AS FOLLOWS :   1  0   0  0  0  1  1  0  0  0  1  1
THIS COVERAGE RESULTED WITH     KSET=  57          KRT=  72        YUK=    4
------------------------------------------------------------------------------
DELETE PATH :  70(  62)  PLANES USED= 3  LOADS COVERED=  3
PLANE ASSIGNMENTS:  72,  71,  66,
LOAD COVERAGE IS AS FOLLOWS :   0  0   0  0  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH     KSET=  61          KRT=  58        YUK=    3
------------------------------------------------------------------------------
INCLUDE PATH :  62(  61)  PLANES USED= 4  LOADS COVERED=  5
PLANE ASSIGNMENTS:  72,  71,  66,  61,
LOAD COVERAGE IS AS FOLLOWS :   1  0   0  0  0  1  0  1  0  0  1  1
THIS COVERAGE RESULTED WITH     KSET=  57          KRT=  72        YUK=    4
------------------------------------------------------------------------------
DELETE PATH :  62(  61)  PLANES USED= 3  LOADS COVERED=  3
PLANE ASSIGNMENTS:  72,  71,  66,
LOAD COVERAGE IS AS FOLLOWS :   0  0   0  0  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH     KSET=  60          KRT=  58        YUK=    3
------------------------------------------------------------------------------
INCLUDE PATH :  45(  60)  PLANES USED= 4  LOADS COVERED=  4
PLANE ASSIGNMENTS:  72,  71,  66,  60,
LOAD COVERAGE IS AS FOLLOWS :   1  0   0  0  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH     KSET=  57          KRT=  72        YUK=    4
------------------------------------------------------------------------------
DELETE PATH :  45(  60)  PLANES USED= 3  LOADS COVERED=  3
PLANE ASSIGNMENTS:  72,  71,  66,
LOAD COVERAGE IS AS FOLLOWS :   0  0   0  0  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH     KSET=  59          KRT=  58        YUK=    3
------------------------------------------------------------------------------
INCLUDE PATH :  35(  59)  PLANES USED= 4  LOADS COVERED=  5
PLANE ASSIGNMENTS:  72,  71,  66,  59,
LOAD COVERAGE IS AS FOLLOWS :   1  1   0  0  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH     KSET=  57          KRT=  72        YUK=    4
------------------------------------------------------------------------------
DELETE PATH :  35(  59)  PLANES USED= 3  LOADS COVERED=  3
PLANE ASSIGNMENTS:  72,  71,  66,
LOAD COVERAGE IS AS FOLLOWS :   0  0   0  0  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH     KSET=  58          KRT=  58        YUK=    3
------------------------------------------------------------------------------
INCLUDE PATH :  23(  58)  PLANES USED= 4  LOADS COVERED=  4
PLANE ASSIGNMENTS:  72,  71,  66,  58,
LOAD COVERAGE IS AS FOLLOWS :   1  0   0  0  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH     KSET=  57          KRT=  72        YUK=    4
------------------------------------------------------------------------------
DELETE PATH :  23(  58)  PLANES USED= 3  LOADS COVERED=  3
PLANE ASSIGNMENTS:  72,  71,  66,
LOAD COVERAGE IS AS FOLLOWS :   0  0   0  0  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH     KSET=  57          KRT=  58        YUK=    3
------------------------------------------------------------------------------
DELETE PATH :  34(  66)  PLANES USED= 2  LOADS COVERED=  2
PLANE ASSIGNMENTS:  72,  71,
LOAD COVERAGE IS AS FOLLOWS :   0  0   0  0  0  0  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH     KSET=  65          KRT=  65        YUK=    3
------------------------------------------------------------------------------
```

```
INCLUDE PATH :  19(  65)  PLANES USED= 3  LOADS COVERED=  3
PLANE ASSIGNMENTS:  72,  71,  65,
LOAD COVERAGE IS AS FOLLOWS :   0  0  0  0  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  64         KRT=  58         YUK=   3
--------------------------------------------------------------------------
INCLUDE PATH :  35(  59)  PLANES USED= 4  LOADS COVERED=  5
PLANE ASSIGNMENTS:  72,  71,  65,  59,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  0  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  57         KRT=  72         YUK=   4
--------------------------------------------------------------------------
DELETE PATH :  35(  59)  PLANES USED= 3  LOADS COVERED=  3
PLANE ASSIGNMENTS:  72,  71,  65,
LOAD COVERAGE IS AS FOLLOWS :   0  0  0  0  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  58         KRT=  58         YUK=   3
--------------------------------------------------------------------------
INCLUDE PATH :  23(  58)  PLANES USED= 4  LOADS COVERED=  4
PLANE ASSIGNMENTS:  72,  71,  65,  58,
LOAD COVERAGE IS AS FOLLOWS :   1  0  0  0  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  57         KRT=  72         YUK=   4
--------------------------------------------------------------------------
DELETE PATH :  23(  58)  PLANES USED= 3  LOADS COVERED=  3
PLANE ASSIGNMENTS:  72,  71,  65,
LOAD COVERAGE IS AS FOLLOWS :   0  0  0  0  0  1  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  57         KRT=  58         YUK=   3
--------------------------------------------------------------------------
DELETE PATH :  19(  65)  PLANES USED= 2  LOADS COVERED=  2
PLANE ASSIGNMENTS:  72,  71,
LOAD COVERAGE IS AS FOLLOWS :   0  0  0  0  0  0  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  54         KRT=  65         YUK=   3
--------------------------------------------------------------------------
DELETE PATH :  69(  71)  PLANES USED= 1  LOADS COVERED=  1
PLANE ASSIGNMENTS:  72,
LOAD COVERAGE IS AS FOLLOWS :   0  0  0  0  0  0  0  0  0  0  1  0
THIS COVERAGE RESULTED WITH    KSET=  70         KRT=  69         YUK=   3
--------------------------------------------------------------------------
INCLUDE PATH :  44(  70)  PLANES USED= 2  LOADS COVERED=  3
PLANE ASSIGNMENTS:  72,  70,
LOAD COVERAGE IS AS FOLLOWS :   0  1  0  0  0  0  0  0  0  0  1  1
THIS COVERAGE RESULTED WITH    KSET=  58         KRT=  65         YUK=   3
--------------------------------------------------------------------------
INCLUDE PATH :  58(  68)  PLANES USED= 3  LOADS COVERED=  5
PLANE ASSIGNMENTS:  72,  70,  68,
LOAD COVERAGE IS AS FOLLOWS :   0  1  0  0  0  1  0  0  0  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  64         KRT=  58         YUK=   3
--------------------------------------------------------------------------
INCLUDE PATH :  72(  64)  PLANES USED= 4  LOADS COVERED=  7
PLANE ASSIGNMENTS:  72,  70,  68,  64,
LOAD COVERAGE IS AS FOLLOWS :   1  1  0  0  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  57         KRT=  32         YUK=   3
--------------------------------------------------------------------------
INCLUDE PATH :  67(  42)  PLANES USED= 5  LOADS COVERED=  9
PLANE ASSIGNMENTS:  72,  70,  68,  64,  42,
LOAD COVERAGE IS AS FOLLOWS :   1  1  1  0  1  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  31         KRT=  14         YUK=   3
--------------------------------------------------------------------------
INCLUDE PATH :  66(  23)  PLANES USED= 6  LOADS COVERED= 11
PLANE ASSIGNMENTS:  72,  70,  68,  64,  42,  23,
LOAD COVERAGE IS AS FOLLOWS :   1  1  1  0  1  1  1  1  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  13         KRT=   7         YUK=   1
--------------------------------------------------------------------------
DELETE PATH :  66(  23)  PLANES USED= 5  LOADS COVERED=  9
PLANE ASSIGNMENTS:  72,  70,  68,  64,  42,
LOAD COVERAGE IS AS FOLLOWS :   1  1  1  0  1  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  22         KRT=  14         YUK=   3
--------------------------------------------------------------------------
INCLUDE PATH :  20(  16)  PLANES USED= 6  LOADS COVERED= 11
PLANE ASSIGNMENTS:  72,  70,  68,  64,  42,  16,
LOAD COVERAGE IS AS FOLLOWS :   1  1  1  1  1  1  0  1  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  13         KRT=   1         YUK=   1
--------------------------------------------------------------------------
DELETE PATH :  20(  16)  PLANES USED= 5  LOADS COVERED=  9
PLANE ASSIGNMENTS:  72,  70,  68,  64,  42,
LOAD COVERAGE IS AS FOLLOWS :   1  1  1  0  1  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  15         KRT=  14         YUK=   3
--------------------------------------------------------------------------
INCLUDE PATH :  10(  15)  PLANES USED= 6  LOADS COVERED= 10
PLANE ASSIGNMENTS:  72,  70,  68,  64,  42,  15,
LOAD COVERAGE IS AS FOLLOWS :   1  1  1  0  1  1  0  1  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  13         KRT=   7         YUK=   1
--------------------------------------------------------------------------
DELETE PATH :  10(  15)  PLANES USED= 5  LOADS COVERED=  9
PLANE ASSIGNMENTS:  72,  70,  68,  64,  42,
LOAD COVERAGE IS AS FOLLOWS :    1  1  1  0  1  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  14        KRT=  14       YUK=   3
----------------------------------------------------------------------
DELETE PATH :  67(  42)  PLANES USED= 4  LOADS COVERED=  7
PLANE ASSIGNMENTS:  72,  70,  68,  64,
LOAD COVERAGE IS AS FOLLOWS :    1  1  0  0  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  41        KRT=  32       YUK=   3
----------------------------------------------------------------------
INCLUDE PATH :  65(  41)  PLANES USED= 5  LOADS COVERED=  9
PLANE ASSIGNMENTS:  72,  70,  68,  64,  41,
LOAD COVERAGE IS AS FOLLOWS :    1  1  1  0  0  1  1  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  31        KRT=  24       YUK=   3
----------------------------------------------------------------------
INCLUDE PATH :  38(  27)  PLANES USED= 6  LOADS COVERED= 11
PLANE ASSIGNMENTS:  72,  70,  68,  64,  41,  27,
LOAD COVERAGE IS AS FOLLOWS :    1  1  1  1  1  1  1  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  23        KRT=  14       YUK=   1
----------------------------------------------------------------------
DELETE PATH :  38(  27)  PLANES USED= 5  LOADS COVERED=  9
PLANE ASSIGNMENTS:  72,  70,  68,  64,  41,
LOAD COVERAGE IS AS FOLLOWS :    1  1  1  0  0  1  1  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  26        KRT=  24       YUK=   3
----------------------------------------------------------------------
INCLUDE PATH :   5(  24)  PLANES USED= 6  LOADS COVERED= 10
PLANE ASSIGNMENTS:  72,  70,  68,  64,  41,  24,
LOAD COVERAGE IS AS FOLLOWS :    1  1  1  0  1  1  1  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  23        KRT=  14       YUK=   1
----------------------------------------------------------------------
DELETE PATH :   5(  24)  PLANES USED= 5  LOADS COVERED=  9
PLANE ASSIGNMENTS:  72,  70,  68,  64,  41,
LOAD COVERAGE IS AS FOLLOWS :    1  1  1  0  0  1  1  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  23        KRT=  24       YUK=   3
----------------------------------------------------------------------
DELETE PATH :  65(  41)  PLANES USED= 4  LOADS COVERED=  7
PLANE ASSIGNMENTS:  72,  70,  68,  64,
LOAD COVERAGE IS AS FOLLOWS :    1  1  0  0  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  40        KRT=  32       YUK=   3
----------------------------------------------------------------------
INCLUDE PATH :  56(  39)  PLANES USED= 5  LOADS COVERED=  9
PLANE ASSIGNMENTS:  72,  70,  68,  64,  39,
LOAD COVERAGE IS AS FOLLOWS :    1  1  1  1  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  31        KRT=  24       YUK=   3
----------------------------------------------------------------------
INCLUDE PATH :  40(  28)  PLANES USED= 6  LOADS COVERED= 11
PLANE ASSIGNMENTS:  72,  70,  68,  64,  39,  28,
LOAD COVERAGE IS AS FOLLOWS :    1  1  1  1  1  1  1  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  23        KRT=  14       YUK=   1
----------------------------------------------------------------------
DELETE PATH :  40(  28)  PLANES USED= 5  LOADS COVERED=  9
PLANE ASSIGNMENTS:  72,  70,  68,  64,  39,
LOAD COVERAGE IS AS FOLLOWS :    1  1  1  1  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  27        KRT=  24       YUK=   3
----------------------------------------------------------------------
INCLUDE PATH :   5(  24)  PLANES USED= 6  LOADS COVERED= 10
PLANE ASSIGNMENTS:  72,  70,  68,  64,  39,  24,
LOAD COVERAGE IS AS FOLLOWS :    1  1  1  1  1  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  23        KRT=  14       YUK=   1
----------------------------------------------------------------------
DELETE PATH :   5(  24)  PLANES USED= 5  LOADS COVERED=  9
PLANE ASSIGNMENTS:  72,  70,  68,  64,  39,
LOAD COVERAGE IS AS FOLLOWS :    1  1  1  1  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  23        KRT=  24       YUK=   3
----------------------------------------------------------------------
DELETE PATH :  56(  39)  PLANES USED= 4  LOADS COVERED=  7
PLANE ASSIGNMENTS:  72,  70,  68,  64,
LOAD COVERAGE IS AS FOLLOWS :    1  1  0  0  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  38        KRT=  32       YUK=   3
----------------------------------------------------------------------
INCLUDE PATH :  55(  38)  PLANES USED= 5  LOADS COVERED=  9
PLANE ASSIGNMENTS:  72,  70,  68,  64,  38,
LOAD COVERAGE IS AS FOLLOWS :    1  1  1  1  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  31        KRT=  24       YUK=   3
----------------------------------------------------------------------
INCLUDE PATH :  25(  26)  PLANES USED= 6  LOADS COVERED= 11
PLANE ASSIGNMENTS:  72,  70,  68,  64,  38,  26,
LOAD COVERAGE IS AS FOLLOWS :    1  1  1  1  1  1  1  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  23        KRT=  14       YUK=   1
----------------------------------------------------------------------
DELETE PATH :  25(  26)  PLANES USED= 5  LOADS COVERED=  9
PLANE ASSIGNMENTS:  72,  70,  68,  64,  38,
LOAD COVERAGE IS AS FOLLOWS :    1  1  1  1  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH     KSET=  25        KRT=  24       YUK=   3
```

```
-----------------------------------------------------------------------------
INCLUDE PATH :  15(  25)  PLANES USED= 6  LOADS COVERED= 10
PLANE ASSIGNMENTS:  72,  70,  68,  64,  38,  25,
LOAD COVERAGE IS AS FOLLOWS :  1  1  1  1  1  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  23       KRT=  14      YUK=   1
-----------------------------------------------------------------------------
DELETE PATH :  15(  25)  PLANES USED= 5  LOADS COVERED=  9
PLANE ASSIGNMENTS:  72,  70,  68,  64,  38,
LOAD COVERAGE IS AS FOLLOWS :  1  1  1  1  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  24       KRT=  24      YUK=   3
-----------------------------------------------------------------------------
DELETE PATH :  55(  38)  PLANES USED= 4  LOADS COVERED=  7
PLANE ASSIGNMENTS:  72,  70,  68,  64,
LOAD COVERAGE IS AS FOLLOWS :  1  1  0  0  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  37       KRT=  32      YUK=   3
-----------------------------------------------------------------------------
INCLUDE PATH :  54(  37)  PLANES USED= 5  LOADS COVERED=  9
PLANE ASSIGNMENTS:  72,  70,  68,  64,  37,
LOAD COVERAGE IS AS FOLLOWS :  1  1  1  0  0  1  0  1  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  31       KRT=  24      YUK=   3
-----------------------------------------------------------------------------
INCLUDE PATH :  25(  26)  PLANES USED= 6  LOADS COVERED= 11
PLANE ASSIGNMENTS:  72,  70,  68,  64,  37,  26,
LOAD COVERAGE IS AS FOLLOWS :  1  1  1  0  1  1  1  1  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  23       KRT=   7      YUK=   1
-----------------------------------------------------------------------------
DELETE PATH :  25(  26)  PLANES USED= 5  LOADS COVERED=  9
PLANE ASSIGNMENTS:  72,  70,  68,  64,  37,
LOAD COVERAGE IS AS FOLLOWS :  1  1  1  0  0  1  0  1  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  25       KRT=  24      YUK=   3
-----------------------------------------------------------------------------
INCLUDE PATH :  15(  25)  PLANES USED= 6  LOADS COVERED= 10
PLANE ASSIGNMENTS:  72,  70,  68,  64,  37,  25,
LOAD COVERAGE IS AS FOLLOWS :  1  1  1  0  1  1  0  1  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  23       KRT=   7      YUK=   1
-----------------------------------------------------------------------------
DELETE PATH :  15(  25)  PLANES USED= 5  LOADS COVERED=  9
PLANE ASSIGNMENTS:  72,  70,  68,  64,  37,
LOAD COVERAGE IS AS FOLLOWS :  1  1  1  0  0  1  0  1  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  24       KRT=  24      YUK=   3
-----------------------------------------------------------------------------
DELETE PATH :  54(  37)  PLANES USED= 4  LOADS COVERED=  7
PLANE ASSIGNMENTS:  72,  70,  68,  64,
LOAD COVERAGE IS AS FOLLOWS :  1  1  0  0  0  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  36       KRT=  32      YUK=   3
-----------------------------------------------------------------------------
INCLUDE PATH :  32(  35)  PLANES USED= 5  LOADS COVERED=  9
PLANE ASSIGNMENTS:  72,  70,  68,  64,  35,
LOAD COVERAGE IS AS FOLLOWS :  1  1  1  0  1  1  0  0  1  1  1  1
THIS COVERAGE RESULTED WITH    KSET=  31       KRT=  14      YUK=   3
-----------------------------------------------------------------------------
INCLUDE PATH :  59(  22)  PLANES USED= 6  LOADS COVERED= 12
PLANE ASSIGNMENTS:  72,  70,  68,  64,  35,  22,
LOAD COVERAGE IS AS FOLLOWS :  1  1  1  1  1  1  1  1  1  1  1  1
```

```
MINIMAX OPTIMAL SOLUTION FOUND AND GIVEN BELOW
**********************************************

PLANE NO : 1      TO  RELATIVE PATH :   72    TO PATH :    63   PORT : 4   TIME : 120
LOADS CARRIED :  11
PLANE NO : 2      TO  RELATIVE PATH :   70    TO PATH :    44   PORT : 4   TIME : 110
LOADS CARRIED :   2  12
PLANE NO : 3      TO  RELATIVE PATH :   68    TO PATH :    58   PORT : 4   TIME : 115
LOADS CARRIED :  10   6
PLANE NO : 4      TO  RELATIVE PATH :   64    TO PATH :    72   PORT : 3   TIME : 120
LOADS CARRIED :   9   1
PLANE NO : 5      TO  RELATIVE PATH :   35    TO PATH :    32   PORT : 4   TIME : 100
LOADS CARRIED :   5   3
PLANE NO : 6      TO  RELATIVE PATH :   22    TO PATH :    59   PORT : 3   TIME : 115
LOADS CARRIED :   7   4   8

MINIMAX OPTIMAL SOLUTION VALUE IS :   120

TOTAL MISSION TIME :  680



ΔBRKPT   PRINT$
```

APPENDIX E

```
***********************************************************************
*                                                                     *
*                                                                     *
*                              PROGRAMS                               *
*                                                                     *
*                                 OF                                  *
*                                                                     *
*      ROUTING  AND  SCHEDULING OF MILITARY CARGO AIRCRAFT            *
*                                                                     *
*                                                                     *
***********************************************************************


        PREPARED BY   : M U R A T    K A S A R O G L U
        ***************************************************
```

```
PROGRAMS OF ,BOTTLENECK ROUTING OF MILITARY CARGO AIRCRAFT,
-----------------------------------------------------------

PREPARED BY : MURAT KASAROGLU

DEFINITIONS OF PROGRAM LIMITS :
-------------------------------

PMAX   =MAXIMUM NUMBER OF PORTS

PLMAX  =MAXIMUM NUMBER OF PLANES

FFMAX  =MAXIMUM NUMBER OF PATHS TO BE GENERATED

NBTN   =MAXIMUM NUMBER OF PATHS THAT ROUTING PROBLEMS CAN HANDLE

MMAX   =MAXIMUM NUMBER OF ELEMENTS PER PATH

FMAX   =MAXIMUM SIZE OF W

LMAX   =MAXIMUM NUMBER OF LOADS

DEFINITIONS OF VARIABLES USED IN PROGRAMS AND INPUT DATA
--------------------------------------------------------

DMAX   =MAXIMUM LENGHT OF PATHS TO BE GENERATED

NPORT  =NUMBER OF PORTS

NLOAD  =NUMBER OF LOADS

NPLANE=NUMBER OF PLANES

NPP    =NUMBER OF PORTS WITH PLANES INITIALLY

NN     =NPP+NLOAD

NNODE  =NN+2
       :TOTAL NUMBER OF NODES ON FORMULATED W

S      =DUMMY SOURCE NODE TO GENERATE THE PATHS

T      =DUMMY TERMINAL NODE TO GENERATE THE PATHS

D      =PORT TO PORT FLIGHT TIME MATRIX
       :D(I,J)=FLIGHT TIME BETWEEN PORT I AND PORT J

LOAD   =LOADS BETWEEN PORTS
       :LOAD(I,1)=STARTING PORT OF I,TH LOAD
       :LOAD(I,2)=ENDING   PORT OF I,TH LOAD

MAIR   =NUMBER OF PLANES AT EACH PORT
       :MAIR(I)=NUMBER OF PLANES AT PORT I

TLOAD  =TIME OF LOADING AND UNLOADING

W      =FORMULATED MATRIX ON WHICH PATHS ARE GENERATED

KTH    =TOTAL NUMBER OF PATHS GENERATED

MC     =TOTAL NUMBER OF PATHS AFTER REMOVAL OF REPETITIONS

PATH   =CONTAINS THE PATH CURRENTLY READ

PLEN   =TIME LENGHT OF A PATH

LBOY   =NUMBER OF ELEMENTS ON A PATH

FRE    =THE FREQUENCY MATRIX
       :FRE(I,J)=NUMBER OF TIMES NODE J OCCURRED
                 ON A I MEMBERED GROUP   (J > 1)
       :FRE(I,J)=NUMBER OF TIMES I MEMEBERED GROUP OCCURRED (J = 1)

FIRST  =THE FIRST OCCURRANCE MATRIX
```

```
          :FIRST(I,J)=PATH NUMBER OF FIRST OCCURANCE OF NODE J
                       ON A I MEMBERED GROUP (J > 1)
          :FIRST(I,J)=PATH NUMBER OF FIRST OCCURANCE OF A
                      I MEMBERED GROUP  (J = 1)

ADRES =THE ADRES ARRAY FOR HANDLING BLOCKING

YFIRST=FIRST OCCURRANCE OF LOADS AFTER BLOCKING
      :YFIRST(J)=FIRST OCCURRANCE OF LOAD J AFTER BLOCKING

SFIRST=FIRST OCCURRANCE OF GROUPS AFTER BLOCKING
      :SFIRST(I)=FIRST OCCURRANCE OF GROUP I AFTER BLOCKING

OPT   =CURRENT PARTIAL SOLUTION GENERATED

ADJ   =ARRAY THAT STORES THE PATHS TO BE PROCESSED

DIS   =ARRAY THAT STORES THE LENGHTS OF PATHS

KLEM  =ARRAY THAT STORES THE NUMBER OF ELEMENTS OF PATHS

AIR   =ARRAY THAT STORES THE ORIGINAL PORTS OF PATHS


FILES USED WITHIN THE PROGRAMS :
--------------------------------

MTAPE  =STORES ALL THE PATHS GENERATED
        (RANDOM ACCESS)

LTAPE  =STORES THE PATHS THAT ARE PASSED REMOVAL
        (SEQUENTIAL ACCESS)

MTAPE  =STORES PATHS THAT ARE SORTED ACCORDING TO INCREASING
        LENGHT SO TO BE PROCESSED BY ROUTING PROGRAMS
        (SEQUENTIAL ACCESS)


PROGRAMS RELATED WITH BOTTLENECK ROUTING :
------------------------------------------

THESIS-1 : GENERATE AND/OR REMOVE PATHS ON HARD DISK

THESIS-3 : REMOVE PATHS ON MEMORY

THESIS-4 : SORT PATHS ON MEMORY

THESIS-5 : SORT PATHS ON HARD DISK

THESIS-6 : ROUTE PLANES USING ,METHOD-A, FOR BLOCKING

THESIS-7 : ROUTE PLANES USING ,METHOD-B, FOR BLOCKING

THESIS-8 : SCHEDULE PLANES SUCH THAT THE PORT CAPACITY CONSTRAINTS D

THESIS-12: ROUTE PLANES USING ,SPP ALGORITHM,


HESIS-1
```

```
C
C     PROGRAM TO GENERATE
C     AND/OR
C     TO REMOVE PATHS ON HARD DISK
C
C     ****************************
C
C     DEFITIONS OF PROGRAM VARIABLES :
C     ********************************
C
C     DEMAND=DEMANDS OF NODES IN ORDER TO HELP REMOVING PATHS
C
C     PDEM  =DEMANDS OF PATHS
C           :SUM DEMANDS OF NODES ON EACH PATH
C
C     CONT  =DEMAND CONTROL ARRAY
C           :CONT(I)=FIRST OCCURRANCE OF A PATH WITH DEMAND OF I
C
C     TDMAX =MAXIMUM PATH DEMAND
C
C
C     ****************************************************************
C
      IMPLICIT INTEGER (A-Z)
      PARAMETER MTAPE=7
      PARAMETER LTAPE=8
      PARAMETER PMAX=30
      PARAMETER PLMAX=30
      PARAMETER LMAX=100
      PARAMETER FMAX=100
      PARAMETER FFMAX=FMAX*FMAX
      PARAMETER TDMAX=1000
      PARAMETER MMAX=8
      PARAMETER HESAP=LMAX+PMAX+2
      DEFINE FILE MTAPE(FFMAX,22,U,KLM)
      DIMENSION D(PMAX,PMAX)
      DIMENSION LOAD(LMAX,2)
      DIMENSION MAIR(PMAX)
      DIMENSION W(FMAX,FMAX)
      DIMENSION DEMAND(FMAX)
      DIMENSION BOS(FMAX)
      DIMENSION PATH(FMAX)
      DIMENSION CONT(TDMAX)
      DIMENSION PDEM(FFMAX)
      DIMENSION CON1(FMAX),CON2(FMAX)
      EQUIVALENCE (W(1,1),PDEM(1))
      PRINT 15000
15000 FORMAT(1H1)
      A=999999
      REMOVE=0
C     READ(5,*)REMOVE
C     GET THE INPUT DATA
C     ******************
      READ(5,*)NPORT
      DO 51 I=1,NPORT
   51 READ(5,*)(D(I,J),J=1,NPORT)
      READ(5,*)(MAIR(I),I=1,NPORT)
      READ(5,*)TLOAD
      I=0
   55 I=I+1
      READ(5,*,END=59,ERR=59)LOAD(I,1),LOAD(I,2)
      GO TO 55
   59 NLOAD=I-1
      IF(NLOAD.GT.LMAX)THEN
      WRITE(6,28000)NLOAD
28000 FORMAT(/,10X,'NUMBER OF LOADS IS',I4,' EXCEEDS LIMITS')
      STOP
      END IF
      READ(5,*)DMAX
C     FORMULATION
C     ***********
      NAIR=0
      DO 52 I=1,NPORT
   52 NAIR=NAIR+MAIR(I)
      NR,NC=NPORT
      WRITE(6,1000)NPORT
 1000 FORMAT(/,10X,'NUMBER OF PORTS       :',I4)
      WRITE(6,2000)NLOAD
 2000 FORMAT(/,10X,'NUMBER OF LOADS       :',I4)
      WRITE(6,3000)TLOAD
```

```
3000 FORMAT(/,10X,,LOADING PLUS UNLOADING TIME :,,I4)
     WRITE(6,4000)
4000 FORMAT(/,10X,,AVAILABLE PLANES AT AIRPORTS,,/,10X,28(,*,))
     DO 54 I=1,NPORT
     WRITE(6,5000)I,MAIR(I)
5000 FORMAT(10X,,PORT,,I4,10X,,AVAILABLE PLANES,,I5)
  54 CONTINUE
     NPP,NPLANE=0
     DO 61 I=1,NPORT
     IF(MAIR(I).GT.0)THEN
     NPP=NPP+1
     NPLANE=NPLANE+MAIR(I)
     END IF
  61 CONTINUE
     IF(NPLANE.GT.PLMAX)THEN
     WRITE(6,27000)NPLANE
7000 FORMAT(/,10X,,NUMBER OF PLANES IS,,I4,, EXCEEDS LIMITS,)
     STOP
     END IF
     NN=NPP+NLOAD
     NNODE=NN+2
     WRITE(6,19000)NPLANE
9000 FORMAT(/,10X,,TOTAL NUMBER OF PLANES :,,I4)
     WRITE(6,11000)
1000 FORMAT(/,10X,,DEMANDS OF NODES,,/,10X,16(,*,),/)
     DO 101 I=1,NNODE
     DEMAND(I)=I+10
     WRITE(6,10000)I,DEMAND(I)
0000 FORMAT(10X,,NODE,,I4,,  DEMAND,,I4)
 101 CONTINUE
     S=1
     T=NN+2
     NK=NPP+1
     NOKTA=1
     DO 62 I=1,NNODE
     DO 62 J=1,NNODE
  62 W(I,J)=A
     DO 72 I=1,NPORT
     IF(MAIR(I).EQ.0)GO TO 72
     NOKTA=NOKTA+1
     W(S,NOKTA)=1
     DO 76 K=1,NLOAD
     W(NOKTA,NK+K)=D(LOAD(K,1),LOAD(K,2))+TLOAD
     IF(LOAD(K,1).NE.I)THEN
     W(NOKTA,NK+K)=W(NOKTA,NK+K)+D(I,LOAD(K,1))
     END IF
  76 CONTINUE
  72 CONTINUE
     DO 75 I=1,NLOAD
     NOKTA=NOKTA+1
  75 W(NOKTA,T)=1
     DO 77 K=1,NLOAD
     DO 78 L=1,NLOAD
     IF(K.EQ.L)GO TO 78
     W(NK+K,NK+L)=D(LOAD(L,1),LOAD(L,2))+TLOAD
     IF(LOAD(K,2).NE.LOAD(L,1))THEN
     W(NK+K,NK+L)=W(NK+K,NK+L)+D(LOAD(K,2),LOAD(L,1))
     END IF
  78 CONTINUE
  77 CONTINUE
     WRITE(6,6000)DMAX
6000 FORMAT(/,10X,,MAXIMUM DISTANCE PERMITTED TO GENERATE PATHS :,,I4)
     WRITE(6,7000)NNODE
7000 FORMAT(/,10X,,TOTAL NUMBER OF NODES GENERATED :,,I4)
     WRITE(6,8000)
8000 FORMAT(/,10X,,MEANINGS OF NODES GENERATED,,/,10X,27(,*,),/)
     WRITE(6,8100)S
8100 FORMAT(10X,,NODE,,I4,, IS THE DUMMY SOURCE NODE,)
     NOKTA=1
     DO 95 I=1,NPORT
     IF(MAIR(I).EQ.0)GO TO 95
     NOKTA=NOKTA+1
     WRITE(6,8200)NOKTA,I
8200 FORMAT(10X,,NODE,,I4,, IS THE PORT,,I4)
  95 CONTINUE
     DO 96 I=1,NLOAD
     NOKTA=NOKTA+1
     WRITE(6,8300)NOKTA,LOAD(I,1),LOAD(I,2)
8300 FORMAT(10X,,NODE,,I4,, IS THE LOAD FROM PORT,,I4,,  TO PORT,,I4)
  96 CONTINUE
     WRITE(6,8400)T
8400 FORMAT(10X,,NODE,,I4,, IS THE DUMMY TERMINAL NODE,)
```

```
      NR,NC=NNODE
      GENERATE PATHS
      **************
      DDMAX=DMAX+2
      CALL ROUTS(W,S,T,NNODE,DDMAX,FMAX,BOS,PATH,FFMAX,MTAPE,MMAX,
     &   A,KTH)
      WRITE(6,7000)NNODE
      WRITE(6,9000)KTH
9000  FORMAT(/,10X,,TOTAL NUMBER OF PATHS GENERATED :,,I10)
      REMOVE REPEATED PATHS
      IF(REMOVE.EQ.0)STOP
      *********************
      OKJ=0
      DO 126 I=1,TDMAX
 126  CONT(I)=0
      MC=0
      DO 121 P=1,KTH
      RK=P/10
      IF(RK*10.EQ.P)THEN
      WRITE(6,788)P
788   FORMAT(/,10X,,PROCESSING RECORD,,I8)
      WRITE(6,789)OKJ
789   FORMAT(10X,,NUMBER OF RECORS READ TILL HERE,,I8)
      END IF
      MC=MC+1
      FIND(MTAPE,MC)
      READ(MTAPE,MC)LBOY,LEN,(BOS(I),I=1,LBOY)
 700  FORMAT(2I8,20I4)
      OKJ=OKJ+1
      PD=0
      DO 122 I=1,LBOY
      PD=PD+DEMAND(BOS(I))
 122  CONTINUE
      TL=PD
      IF(PD.GT.TDMAX)TL=TDMAX
      PDEM(P)=PD
      IF(CONT(TL).EQ.0)THEN
      CONT(TL)=P
      GO TO 121
      END IF
      DO 127 I=1,NNODE
 127  CON1(I)=0
      DO 128 I=1,LBOY
 128  CON1(BOS(I))=1
      DO 123 PP=CONT(TL),P-1
      IF(PDEM(PP).LT.0)GO TO 123
      IF(PDEM(PP).NE.PD)GO TO 123
      PPP=PP
      FIND(MTAPE,PPP)
      READ(MTAPE,PPP)PBOY,PEN,(PATH(I),I=1,PBOY)
      OKJ=OKJ+1
      IF(LBOY.NE.PBOY)GO TO 123
      IF(PATH(1).NE.BOS(1))GO TO 123
      DO 131 I=1,NNODE
 131  CON2(I)=0
      DO 132 I=1,PBOY
 132  CON2(PATH(I))=1
      DO 124 I=1,NNODE
      IF(CON1(I).EQ.CON2(I))GO TO 124
      GO TO 123
 124  CONTINUE
      IF(LEN.LT.PEN)THEN
      PDEM(PP)=-PD
      ELSE
      PDEM(P)=-PD
      END IF
      GO TO 121
 123  CONTINUE
 121  CONTINUE
      REWIND LTAPE
      WRITE(6,1660)OKJ
1660  FORMAT(/,10X,,READ PHASE OF PATH REDUCTION HAS ENDED,,/,10X,
     & ,NUMBER OF RECORDS READ,,I8)
      MC=0
      DO 125 P=1,KTH
      IF(PDEM(P).LT.0)GO TO 125
      MC=MC+1
      PP=P
      READ(MTAPE,PP)LBOY,LEN,(BOS(I),I=1,LBOY)
      WRITE(LTAPE,700)LBOY,LEN,(BOS(I),I=1,LBOY)
 125  CONTINUE
      WRITE(6,7000)NNODE
      WRITE(6,12000)MC
12000 FORMAT(/,10X,,NUMBER OF REDUCED PATHS,,I8)
      END FILE LTAPE
      STOP
      END
.THESIS-2
```

```
      SUBROUTINE ROUT5(DA,S,T,NNODE,LMAX,NNODES,PATH,MARK,FFMAX,MTAPE,
     & MMAX,
     & AAA,KTH)
C
C     *****************************************
C     THIS SUBROUTINE FINDS OUT ALL SIMPLE PATHS IN
C     IN AN UNDIRECTED OR DIRECTED GRAPH
C     USING THE WEIGHTED ADJACENCY MATRIX
C     SUBJECT TO MAXIMUM DISTANCE CONSTRAINT
C     *****************************************
C
      IMPLICIT INTEGER(A-Z)
      DIMENSION DA(NNODES,NNODES)
      DIMENSION PATH(NNODES)
      DIMENSION MARK(NNODES)
      DIMENSION A(2)
C     ******
C     STEP 1
C     ******
      LEN=0
      VA=S
      VZ=T
      KTH=0
      BBB=AAA-1
      DO 9 I=1,NNODE
    9 MARK(I)=1
      DO 1 I=1,NNODE
      IF(DA(VA,I).GT.LMAX)GO TO 1
      IF(DA(VA,I).LE.BBB)THEN
      A(1)=VA
      A(2)=I
      MARK(VA),MARK(VZ)=-1
      LBOY=1
      PATH(LBOY)=VA
      LEN=LEN+DA(VA,I)
      ELEN=LEN
      GO TO 102
      END IF
    1 CONTINUE
C     ******
C     STEP 2
C     ******
  102 CONTINUE
      V2=A(2)
      IF(MARK(V2).EQ.1)THEN
      DO 2 I=1,NNODE
      IF(IABS(DA(V2,I)).LE.BBB)THEN
      ELER=IABS(DA(V2,I))+LEN
      IF(ELER.GT.LMAX)GO TO 2
      ELEN=LEN
      LEN=ELER
      MARK(V2)=-1
      DA(A(1),A(2))=-1*DA(A(1),A(2))
      LBOY=LBOY+1
      PATH(LBOY)=V2
      A(1)=V2
      A(2)=I
      GO TO 102
      END IF
    2 CONTINUE
      GO TO 103
      END IF
      IF(V2.EQ.VZ)THEN
      LBOY=LBOY+1
      PATH(LBOY)=VZ
      KTH=KTH+1
      MBOY=LBOY-2
      MLEN=LEN-2
      IF(MBOY-1.GT.MMAX)THEN
      WRITE(6,832)
  832 FORMAT(/,10X,'NUMBER OF LOADS PER PLANE EXCEEDS PROGRAM',
     & ' LIMITS')
      STOP
      END IF
      WRITE(MTAPE,KTH)MBOY,MLEN,(PATH(I),I=2,LBOY-1)
  700 FORMAT(2I8,20I4)
   84 LBOY=LBOY-1
      LEN=LEN-DA(PATH(LBOY),VZ)
      IF(KTH.GE.FFMAX)THEN
      WRITE(6,831)
  831 FORMAT(/,10X,'NUMBER OF PATHS EXCEEDS PROGRAM LIMITS')
```

```
      STOP
      END IF
      END IF
C     ******
C     STEP 3
C     ******
  103 CONTINUE
      V1=A(1)
      IF(A(2).GE.NNODE)GO TO 105
      DO 3 I=A(2)+1,NNODE
      IF(DA(V1,I).GE.0..AND.DA(V1,I).LE.BBB)THEN
      ELEM=ELEN+DA(V1,I)
      IF(ELEM.GT.LMAX)GOTO 3
      LEN=ELEM
      A(1)=V1
      A(2)=I
      GO TO 102
      END IF
    3 CONTINUE
  105 CONTINUE
      IF(V1.EQ.VA)THEN
      RETURN
      END IF
      DO 4 I=1,NNODE
      IF(DA(I,V1).GE.-BBB.AND.DA(I,V1).LE.0.)THEN
      A(1)=I
      A(2)=V1
      MARK(V1)=1
      DA(A(1),A(2))=IABS(DA(A(1),A(2)))
      LBOY=LBOY-1
      IF(LBOY.LE.1)THEN
      LEN=0
      ELEN=0
      ELSE
      LEN=0
      DO 145 IKJ=2,LBOY
      LEN=LEN+IABS(DA(PATH(IKJ-1),PATH(IKJ)))
  145 CONTINUE
      ELEN=LEN
      END IF
      GO TO 103
      END IF
    4 CONTINUE
      END
THESIS-3
```

```
      REMOVE REDUNDANT PATHS ON MEMORY
      ********************************

      DEFITIONS OF PROGRAM VARIABLES :
      ********************************

      DEMAND=DEMANDS OF NODES IN ORDER TO HELP REMOVING PATHS

      PDEM  =DEMANDS OF PATHS
            :SUM DEMANDS OF NODES ON EACH PATH

      CONT  =DEMAND CONTROL ARRAY
            :CONT(I)=FIRST OCCURRANCE OF A PATH WITH DEMAND OF I

      TDMAX =MAXIMUM PATH DEMAND

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

      IMPLICIT INTEGER (A-Z)
      PARAMETER MTAPE=7
      PARAMETER LTAPE=6
      PARAMETER MMAX=8
      PARAMETER FMAX=100
      PARAMETER NBIN=1000
      PARAMETER TDMAX=1000
      PARAMETER FFMAX=FMAX*FMAX
      DEFINE FILE MTAPE(FFMAX,22,U,KLM)
      DIMENSION ADJ(NBIN,MMAX)
      DIMENSION AIR(NBIN)
      DIMENSION DIS(NBIN)
      DIMENSION KLEM(NBIN)
      DIMENSION PATH(MMAX)
      DIMENSION PDEM(NBIN)
      DIMENSION NDEM(FMAX)
      DIMENSION CONT(TDMAX)
      DIMENSION CONJ(FMAX)
      READ(5,*)KTH,NNODE
      DO 301 J=1,FMAX
301   NDEM(J)=J+10
      MC=0
      DO 125 P=1,KTH
      PP=P
      READ(MTAPE,PP)LBOY,LEN,MAIR,(PATH(J),J=1,LBOY-1)
      SUM=NDEM(MAIR)
      DO 201 J=1,LBOY-1
201   SUM=SUM+NDEM(PATH(J))
      SSUM=SUM
      IF(SUM.GT.TDMAX)SUM=TDMAX
      IF(CONT(SUM).EQ.0)THEN
      CONT(SUM)=MC+1
      GO TO 250
      END IF
      DO 150 PPP=CONT(SUM),MC
      IF(PDEM(PPP).NE.SSUM)GO TO 150
      IF(AIR(PPP).NE.MAIR)GO TO 150
      IF(KLEM(PPP).NE.LBOY)GO TO 150
      DO 174 J=1,NNODE
174   CONJ(J)=0
      DO 175 J=1,LBOY-1
175   CONJ(PATH(J))=1
      DO 176 J=1,LBOY-1
      IF(CONJ(ADJ(PPP,J)).NE.1)GO TO 150
176   CONTINUE
      IF(LEN.GE.DIS(PPP))GO TO 125
      DO 177 J=1,LBOY-1
177   ADJ(PPP,J)=PATH(J)
      DIS(PPP)=LEN
      GO TO 125
150   CONTINUE
250   CONTINUE
      MC=MC+1
      IF(MC.GT.NBIN)THEN
      WRITE(5,401)
401   FORMAT(/,10X,'NUMBER OF REDUCED PATHS EXCEEDS PROGRAM LIMITS')
      STOP
      END IF
      DO 251 J=1,LBOY-1
  251 ADJ(MC,J)=PATH(J)
      DIS(MC)=LEN
      AIR(MC)=MAIR
      KLEM(MC)=LBOY
      PDEM(MC)=SSUM
  125 CONTINUE
      DO 275 I=1,MC
      WRITE(LTAPE,700)KLEM(I),DIS(I),AIR(I),(ADJ(I,J),
     & J=1,KLEM(I)-1)
  275 CONTINUE
      WRITE(6,302)MC
  302 FORMAT(//,10X,,NUMBER OF REDUCED PATHS :,,I8)
  700 FORMAT(2I8,20I4)
      STOP
      END
THESIS-4
```

```
C     SORT THE REDUCED PATHS ON MEMORY
C     BY USING HEAP SORT
C     ******************************
C
      IMPLICIT INTEGER (A-Z)
      PARAMETER LTAPE=5
      PARAMETER KTAPE=9
      PARAMETER NBIN=1200
      PARAMETER MMAX=8
      DIMENSION ADJ(NBIN,MMAX)
      DIMENSION AIR(NBIN)
      DIMENSION DIS(NBIN)
      DIMENSION KLEM(NBIN)
      DIMENSION PATH(MMAX)
C     READ THE INPUT FILE
C     *******************
      I=0
 150  I=I+1
      READ(LTAPE,700,END=99,ERR=99)KLEM(I),DIS(I),AIR(I),(ADJ(I,J),
     & J=1,KLEM(I)-1)
 700  FORMAT(2I8,20I4)
      GO TO 150
  99  CONTINUE
      MC=I-1
      N=MC
C     HEAP SORT
C     *********
1730  L=N/2+1
      R=N
1740  IF(R.LT.2)GO TO 250
1750  IF(L.GT.1)THEN
      L=L-1
      J=L
      GO TO 1770
      END IF
1760  DUM=DIS(1)
      DIS(1)=DIS(R)
      DIS(R)=DUM
      DUM=AIR(1)
      AIR(1)=AIR(R)
      AIR(R)=DUM
      DUM=KLEM(1)
      KLEM(1)=KLEM(R)
      KLEM(R)=DUM
      DO 1991 II=1,MMAX
1991  PATH(II)=ADJ(1,II)
      DO 1992 II=1,MMAX
1992  ADJ(1,II)=ADJ(R,II)
      DO 1993 II=1,MMAX
1993  ADJ(R,II)=PATH(II)
      R=R-1
      J=1
1770  IF(2*J.GT.R)GO TO 1740
1780  K=2*J
1790  IF(K.LT.R.AND.DIS(K).LT.DIS(K+1))K=K+1
1810  IF(DIS(J).LT.DIS(K))THEN
      DUM=DIS(J)
      DIS(J)=DIS(K)
      DIS(K)=DUM
      DUM=AIR(J)
      AIR(J)=AIR(K)
      AIR(K)=DUM
      DUM=KLEM(J)
      KLEM(J)=KLEM(K)
      KLEM(K)=DUM
      DO 1994 II=1,MMAX
1994  PATH(II)=ADJ(J,II)
      DO 1995 II=1,MMAX
1995  ADJ(J,II)=ADJ(K,II)
      DO 1996 II=1,MMAX
1996  ADJ(K,II)=PATH(II)
      J=K
      GO TO 1770
      END IF
1820  GOTO 1740
C     WRITE ON OUTPUT FILE
C     ********************
 250  CONTINUE
```

```
      DO 1997 I=1,MC
      WRITE(KTAPE,700)KLEM(I),DIS(I),AIR(I),(ADJ(I,J),
    & J=1,KLEM(I)-1)
 1997 CONTINUE
      STOP
      END
THESIS-5
```

```
C
C
C      SORT THE PATHS ON HARD DISK
C      ***************************
C
C
       COMMON /COM/X(1000)
       MTAPE=8
       LTAPE=9
       REWIND MTAPE
       CALL FSORT(,KEY=9/8/A/S,RSZ=96,CORE=10000B,,MTAPE,LTAPE,X)
       STOP
       END
,THESIS-6
```

```
      *************************************

      SEACH FOR THE BOTTLENECK OPTIMAL
      OF THE ROUTING PROBLEM
      BY USING ,METHOD-A, FOR BLOCKING
      AND
      TRY TO OBTAIN A FEASIBLE SCHEDULE
      UNDER THIS MINIMAX OPTIMAL POINT

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

      IMPLICIT INTEGER (A-Z)
      PARAMETER KTAPE=9
      PARAMETER PMAX=30
      PARAMETER PLMAX=30
      PARAMETER LMAX=100
      PARAMETER FMAX=100
      PARAMETER MMAX=8
      PARAMETER NBIN=1200
      PARAMETER HESAP=LMAX+PMAX+2
      DIMENSION LOAD(LMAX,2)
      DIMENSION MAIR(PMAX)
      DIMENSION PATH(FMAX)
      DIMENSION FRE(MMAX,HESAP)
      DIMENSION FIRST(MMAX,HESAP)
      DIMENSION CON1(FMAX)
      DIMENSION ADJ(NBIN,MMAX)
      DIMENSION KLEM(NBIN)
      DIMENSION AIR(NBIN)
      DIMENSION DIS(NBIN)
      DIMENSION AIRMAX(PMAX)
      DIMENSION AIRMED(PMAX)
      DIMENSION CRT(LMAX)
      DIMENSION OPT(PLMAX)
      DIMENSION COVER(LMAX)
      DIMENSION ADRES(NBIN)
      DIMENSION TOTAL(LMAX)
      DIMENSION CAP(PMAX)
      DIMENSION OCAP(PMAX)
      DIMENSION D(PMAX,PMAX)
      DIMENSION YFIRST(MMAX)
      DIMENSION SFIRST(LMAX)
      COMMON /IMAT1/NR,NC
      CHARACTER AA*50/,***THE FREQUENCY MATRIX***,/
      CHARACTER BB*50/,***THE FIRST OCURRANCE MATRIX***,/
      PRINT 15000
15000 FORMAT(1H1)
      A=999999
      IFEAS=0
      SECTIM=0
      ITIME=A
      DO 1313 JK=1,PMAX
 1313 AIRMAX(JK)=0
      GET THE INPUT DATA
      *******************
      READ(5,*)NPORT
      DO 699 II=1,NPORT
  699 READ(5,*)(D(II,I),I=1,NPORT)
      READ(5,*)(MAIR(I),I=1,NPORT)
      READ(5,*)TLOAD
      I=0
   55 I=I+1
      READ(5,*,END=59,ERR=59)LOAD(I,1),LOAD(I,2)
      GO TO 55
   59 NLOAD=I-1
      READ(5,*)DMAX
      READ(5,*)YUKLE,BOSALT
      READ(5,*)(CAP(I),I=1,NPORT)
      READ(5,*)(OCAP(I),I=1,NPORT)
      IF(TLOAD.NE.YUKLE+BOSALT)THEN
      WRITE(6,7782)
 7782 FORMAT(//,10X,,LOADING , UNLOADING TIME INCONSISTENCY IN DATA,,
     & /,10X,,JOB TERMINATED,)
      STOP
      END IF
      FORMULATION
      ***********
      NAIR=0
      DO 52 I=1,NPORT
   52 NAIR=NAIR+MAIR(I)
      NR,NC=NPORT
      WRITE(6,1570)
 1570 FORMAT(///,10X,'BOTTLENECK ROUTING OF CARGO AIRCRAFT',/,10X,
     & 36('+'),///)
      WRITE(6,1000)NPORT
 1000 FORMAT(/,10X,'NUMBER OF PORTS      :',I4)
      WRITE(6,2000)NLOAD
 2000 FORMAT(/,10X,'NUMBER OF LOADS      :',I4)
      WRITE(6,3000)TLOAD
 3000 FORMAT(/,10X,'LOADING PLUS UNLOADING TIME :',I4)
      WRITE(6,6000)DMAX
 6000 FORMAT(/,10X,'MAXIMUM DISTANCE PERMITTED TO GENERATE PATHS :',I4)
      WRITE(6,7783)YUKLE,BOSALT
 7783 FORMAT(/,10X,'LOADING TIME    :',I6,
     & /,10X,'UNLOADING TIME :',I6)
      WRITE(6,7784)
 7784 FORMAT(/,10X,'PORT CAPACITIES',/,10X,15('*'),/)
      DO 7785 I=1,NPORT
      WRITE(6,7786)I,MAIR(I),CAP(I),QCAP(I)
 7786 FORMAT(10X,'PORT :',I2,'  PLANES :',I2,
     & '  SERVICE CAPACITY :',I2,'  QUE CAPACITY :',I2)
 7785 CONTINUE
      NPP,NPLANE=0
      DO 61 I=1,NPORT
      IF(MATR(I).GT.0)THEN
      NPP=NPP+1
      NPLANE=NPLANE+MATR(I)
      END IF
   61 CONTINUE
      NN=NPP+NLOAD
      NNODE=NN+2
      WRITE(6,9000)NPLANE
 9000 FORMAT(/,10X,'NUMBER OF PLANES :',I4)
      S=1
      T=NN+2
      NK=NPP+1
      WRITE(6,7000)NNODE
 7000 FORMAT(/,10X,'TOTAL NUMBER OF NODES GENERATED :',I4)
      WRITE(6,8000)
 8000 FORMAT(/,10X,'MEANINGS OF NODES GENERATED',/,10X,27('*'),/)
      DO 1564 I=1,NPORT
 1564 CON1(I)=0
      WRITE(6,8100)S
 8100 FORMAT(10X,'NODE',I4,' IS THE DUMMY SOURCE NODE')
      NOKTA=1
      DO 95 I=1,NPORT
      IF(MATR(I).EQ.0)GO TO 95
      NOKTA=NOKTA+1
      WRITE(6,8200)NOKTA,I
      AIRMAX(NOKTA-1)=MAIR(I)
      CON1(NOKTA-1)=I
 8200 FORMAT(10X,'NODE',I4,' IS THE PORT',I4)
   95 CONTINUE
      DO 1565 I=1,NPORT
 1565 MAIR(I)=CON1(I)
      DO 96 I=1,NLOAD
      NOKTA=NOKTA+1
      WRITE(6,8300)NOKTA,LOAD(I,1),LOAD(I,2)
 8300 FORMAT(10X,'NODE',I4,' IS THE LOAD FROM PORT',I4,'  TO PORT',I4)
   96 CONTINUE
      WRITE(6,8400)T
 8400 FORMAT(10X,'NODE',I4,' IS THE DUMMY TERMINAL NODE')
      NR,NC=NNODE
      WRITE(6,1561)
 1561 FORMAT(/,10X,'LOADS BETWEEN PORTS',/,10X,19('*'),/)
      DO 1562 I=1,NLOAD
 1562 WRITE(6,1563)I,LOAD(I,1),LOAD(I,2)
 1563 FORMAT(10X,'LOAD :',I3,'   FROM',I4,'   TO',I4)
      MINIMUM CONFIGURATION SETUP
      ***************************
      WRITE(6,9000)NPLANE
      WRITE(6,2000)NLOAD
      N=NLOAD/NPLANE
      K=NLOAD-N*NPLANE
      PK=NPLANE-K
      M1=N+1
      M2=N+2
      IF(K.EQ.0)THEN
      M1=N
      M2=N+1
      END IF
      MIK=M1*K
      ***************************************
      INFORMATION RETRIVIAL AND PATH SCANNING
      ***************************************
      REWIND KTAPE
      GMAX=0
      ELEN=0
      LS=2+NPP
      LB=1+NPP+NLOAD
      KAIR=NPP
      NRR=NLOAD
      DO 311 I=1,MMAX
      DO 311 J=1,MNODE
      FIRST(I,J),FRE(I,J)=0
  311 CONTINUE
      RBOY=1
      COUNT=0
      WRITE(6,800)
  800 FORMAT(///,10X,'THE LIST OF PATHS GENERATED',/,10X,27('*'),//)
      I=0
  201 I=I+1
      LEN=0
      READ(KTAPE,700,END=999,ERR=999)LBOY,LEN,(PATH(J),J=1,LBOY)
  999 CONTINUE
      IF(LEN.GT.ELEN)THEN
C     FEASIBILITY CHECKS
C     ******************
C     WRITE(6,898)LEN
C 898 FORMAT(/,10X,'FROM HERE ON PATH LENGHTS ARE',I8,/,10X,
C    & 'START CHECKING FOR FEASIBILITY')
      EELEN=ELEN
      ELEN=LEN
      IF(FRE(1,1).LT.NLOAD)THEN
C     WRITE(6,899)FRE(1,1),NLOAD
C 899 FORMAT(/,10X,'NUMBER OF LOADS COVERED IS',I3,/,10X,
C    & 'WHICH IS LESS THEN TOTAL LOADS',I3,/,10X,
C    & 'RETURN FROM FEASIBILITY CHECK',/)
      GO TO 28001
      END IF
      IF(GMAX.LT.M1)THEN
C     WRITE(6,901)GMAX,M1
C 901 FORMAT(/,10X,'NUMBER OF LOADS PER PLANE IS',I3,/,10X,
C    & 'WHICH IS LESS THEN',I3,/,10X,
C    & 'RETURN FROM FEASIBILITY CHECK',/)
      GO TO 28001
      END IF
      IF(GMAX.EQ.M1)THEN
      IF(FRE(M1,1).LT.MIK)THEN
C     WRITE(6,902)FRE(M1,1),M1,MIK
C 902 FORMAT(/,10X,'NUMBER OF DISJOINT NODES',I4,' IN',I3,/,10X,
C    & '             MEMBERED PATHS WHICH IS LESS THEN',I4,/,10X,
C    & 'RETURN FROM FEASIBILITY CHECK',/)
      GO TO 28001
      END IF
C     CHOOSE WORST LOAD
C     *****************
      GMIN=0
      DO 341 KI=LS,LB
      DO 342 G=1,GMAX
      IF(FRE(G,KI).EQ.0)GO TO 315
  342 CONTINUE
      GG=GMAX
      GO TO 325
  315 GG=G-1
  325 IF(GG.LT.GMIN)THEN
      GMIN=GG
      END IF
  341 CONTINUE
C     WRITE(6,801)GMIN
C 801 FORMAT(/,10X,'GMIN IS',I3)
      IF(GMAX.EQ.GMIN)GO TO 28006
      BIG=0
      DO 343 KI=LS,LB
      IF(FRE(GMIN+1,KI).GT.0)GO TO 343
      IF(FRE(GMIN,KI).LT.BIG)THEN
      BIG=FRE(GMIN,KI)
      WORST=KI
      END IF
  343 CONTINUE
C     WRITE(6,812)WORST,BIG
```

```
  812 FORMAT(10X,'WORST=',I3,'          BIG=',I3)
      KALAN=N-GMIN
      IF(KALAN.LE.0)GO TO 28006
      MIKK=M1*(K+KALAN)
      WRITE(6,813)MIKK
  813 FORMAT(10X,'MIKK=',I3)
      IF(FRE(M1,1).LT.MIKK)THEN
      WRITE(6,903)M1,FRE(M1,1),MIKK
  903 FORMAT(/,10X,'NUMBER OF DISJOINT NODES IN',I3,/,10X,
     & '             NUMBERED PATHS IS',I4,'  LESS THEN',I4,/,10X,
     & 'RETURN FROM FEASIBILITY CHECK')
      GO TO 28001
      END IF
      END IF
 8005 CONTINUE
      FIND MINIMUM NUMBER OF PLANES NECCESSARY
      ********************************************
      TPLANE=0
      DO 444 KI=1,NNODE
  444 CON1(KI)=0
      LEFT=0
      DO 441 G=GMAX,1,-1
      LLEFT=0
      DO 442 KI=LS,LB
      IF(FRE(G,KI).EQ.0)GO TO 442
      IF(CON1(KI).EQ.1)GO TO 442
      CON1(KI)=1
      LLEFT=LLEFT+1
  442 CONTINUE
      LEFT=LEFT+LLEFT
      TT=LEFT/G
      TPLANE=TPLANE+TT
      LEFT=LEFT-TT*G
  441 CONTINUE
      WRITE(6,821)TPLANE,NPLANE
  821 FORMAT(/,10X,'AT LEAST',I3,'  PLANE REQUIRED',/,10X,
     & 'ALTHOUGH WE HAVE',I3)
      IF(TPLANE.GT.NPLANE)THEN
      WRITE(6,822)
  822 FORMAT(10X,'MINIMUM PLANE CHECK FAILED',/,10X,
     & 'RETURN FROM FEASIBILITY CHECK')
      GO TO 28001
      END IF
      THERE CAN BE A SOLUTION SEARCH FOR IT
      ************************************
      SEARCH FOR A FEASIBLE SOLUTION
      ******************************
      WRITE(6,7707)
 7707 FORMAT(///,10X,'SEARCHING FOR FEASIBILITY',/,10X,25('*'),//)
      JSET=I-1
      TCOUNT=USET
      WRITE(6,751)LSET,USET
  751 FORMAT(/,10X,'LSET=',I4,'        JSET=',I4)
      BLOCKING OF PATHS
      *****************
      WRITE(6,1505)
 1505 FORMAT(/,10X,'TOTAL FREQUENCIES',/,10X,17('*'),/)
      DO 1501 K=1+NPP+1,NNODE-1
      KM=K-1-NPP
      TOTAL(KM)=0
      DO 1502 KK=1,GMAX
      TOTAL(KM)=TOTAL(KM)+FRE(KK,K)
 1502 CONTINUE
      WRITE(6,1504)KM,TOTAL(KM)
 1504 FORMAT(10X,'LOAD :',I2,'   TOTAL FREQUENCY :',I5)
 1501 CONTINUE
      DO 1509 KKK=1,NLOAD
      TOTMIN=A
      DO 1506 K=1,NLOAD
      IF(TOTAL(K).GE.TOTMIN)GO TO 1506
      TOTMIN=TOTAL(K)
      TOTL=K
 1506 CONTINUE
      DO 1507 K=JSET,FIRST(1,TOTL+1+NPP),-1
      IF(AIR(K).LT.0)GO TO 1507
      DO 1508 KK=1,KLEN(K)
      IF(ADJ(K,KK).NE.TOTL)GO TO 1508
      AIR(K)=-AIR(K)
      ADRES(TCOUNT)=K
      ICOUNT=TCOUNT-1
      GO TO 1507
 1508 CONTINUE
```

```
507 CONTINUE
    TOTAL(TOTL)=A
509 CONTINUE
    DO 1510 K=1,JSET
510 AIR(K)=-AIR(K)
    WRITE(6,1513)
513 FORMAT(/,10X,,THE ADRES TABLE,,/,10X,15(,*,),/)
    DO 1514 K=1,JSET
514 WRITE(6,1515)K,ADRES(K)
515 FORMAT(10X,,NEW CODE :,,I5,,        OLD CODE :,,I5)
    DO 1517 K=1,GMAX
517 YFIRST(K)=0
    IF(GMAX.EQ.MMAX)GO TO 1573
    DO 1530 K=GMAX+1,MMAX
530 YFIRST(K)=JSET
573 DO 1518 K=1,NLOAD
518 SFIRST(K)=0
    DO 1511 K=1,JSET
    KK=ADRES(K)
    KM=KLEM(KK)
    IF(YFIRST(KM).EQ.0)YFIRST(KM)=K
    IF(KM.NE.1)GO TO 1511
    IF(SFIRST(ADJ(KK,1)).EQ.0)SFIRST(ADJ(KK,1))=K
511 CONTINUE
    WRITE(6,1525)
525 FORMAT(/,10X,,RELATIVE FIRST OCCURRANCES OF GROUPS,,/,10X,
  & 36(,*,),/)
    DO 1521 K=1,GMAX
521 WRITE(6,1522)K,YFIRST(K)
522 FORMAT(10X,,GROUP SIZE :,,I2,,   RELATIVE FIRST OCCURRANCE :,,I5)
    WRITE(6,1523)
523 FORMAT(/,10X,,RELATIVE FIRST OCCURRANCES OF LOADS,,/,10X,
  &  35(,*,),/)
    DO 1524 K=1,NLOAD
524 WRITE(6,1525)K,SFIRST(K)
525 FORMAT(10X,,LOAD :,,I2,,   RELATIVE FIRST OCCURRANCE :,,I5)
    DO 909 K=1,NLOAD
909 CRT(K)=SFIRST(K)
    LSET=0
    DO 1519 K=1,NLOAD
    IF(SFIRST(K).GT.LSET)LSET=SFIRST(K)
519 CONTINUE
    NSAYI=JSET-LSET+1
    MSAYI=0
    ITIME=A
    KPL=0
    KLD=0
    DO 701 LL=1,NLOAD
701 COVER(LL)=0
    DO 702 LL=1,NPP
702 AIRMED(LL)=0
    KKSET=JSET
    OPTIMIZE SEARCH SCAN LIMITS
    ***************************
735 CONTINUE
    IF(KPL.EQ.0)THEN
    JTIME=0
    MSAYI=MSAYI+1
    IF(MSAYI.GT.NSAYI)THEN
    WRITE(6,754)
754 FORMAT(/,10X,,NO FEASIBLE SOLUTION CAN BE DETECTED,,/,10X,
  & ,CONTINUE PATH SCANNING.)
    GO TO 745
    END IF
    KSET=KKSET
    YUK=M1
    KKSET=KKSET-1
    KRT=LSET
    ELSE
    REML=NLOAD-KLD
    REMP=NPLANE-KPL
    YUK=REML/REMP
    IF(YUK*REMP.NE.REML)YUK=YUK+1
    KRT=YFIRST(YUK)
    LRT=0
    DO 904 K=1,NLOAD
    IF(CRT(K).GT.LRT)LRT=CRT(K)
904 CONTINUE
    END IF
    IF(LRT.GT.KRT)KRT=LRT
    WRITE(6,7705)KSET,KRT,YUK
7705 FORMAT(10X,,THIS COVERAGE RESULTED WITH     ,,
     & ,KSET=,,I4,,        KRT=,,I4,,      YUK=,,I4,
     & /,10X,50(,-,))
      IF(KSET.GE.KRT)THEN
      GO TO 715
      ELSE
      GO TO 725
      END IF
      SEARCH FOR A FEASIBLE PATH
      **************************
 715  CONTINUE
      DO 703 K=KSET,KRT,-1
      KADR=ADRES(K)
      PORT=AIR(KADR)
      IF(AIRMED(PORT).EQ.AIRMAX(PORT))GO TO 703
      IF(JTIME+DIS(KADR).GE.ITIME)GO TO 703
      DO 704 LL=1,KLEM(KADR)
      NOD=ADJ(KADR,LL)
      IF(COVER(NOD).EQ.1)GO TO 703
 704  CONTINUE
      INCLUDE THE PATH
      ****************
      KPL=KPL+1
      OPT(KPL)=K
      KZ=0
      AIRMED(PORT)=AIRMED(PORT)+1
      JTIME=JTIME+DIS(KADR)
      DO 705 LL=1,KLEM(KADR)
      KLD=KLD+1
      NOD=ADJ(KADR,LL)
      CRT(NOD)=0
      COVER(NOD)=1
      IF(SFIRST(NOD).GT.KZ)KZ=SFIRST(NOD)
 705  CONTINUE
      WRITE(6,7701)KADR,K,KPL,KLD
7701  FORMAT(10X,,INCLUDE PATH :,,I4,,(,,I4,,)  PLANES USED=,,I2,
     & ,  LOADS COVERED=,,I3)
      WRITE(6,7710)(OPT(IKL),IKL=1,KPL)
7710  FORMAT(10X,,PLANE ASSIGNMENTS:,,10(I4,(,,,)))
      WRITE(6,7705)(COVER(IKL),IKL=1,NLOAD)
7705  FORMAT(10X,,LOAD COVERAGE IS AS FOLLOWS : ,,
     &       20(I3))
      IF(KLD.EQ.NLOAD)THEN
      WRITE(6,752)
 752  FORMAT(//////,10X,,MINIMAX OPTIMAL SOLUTION FOUND AND GIVEN BELOW,,
     & /,10X,46(,*,),/)
      TFEAS=TFEAS+1
      ITIME=JTIME
      WRITE(6,1597)TFEAS
1597  FORMAT(/,10X,,ALTERNATE SOLUTION NUMBER :,,I3)
      DO 706 LL=1,KPL
      WRITE(6,753)LL,OPT(LL),ADRES(OPT(LL)),MAIR(AIR(ADRES(OPT(LL)))),
     & DIS(ADRES(OPT(LL)))
 753  FORMAT(/,10X,,PLANE NO :,,I2,,        TO  RELATIVE PATH :,,I6,
     & ,     TO PATH :,,I5,,     PORT :,,I2,,       TIME :,,I6)
      WRITE(6,1567)(ADJ(ADRES(OPT(LL)),JK),JK=1,KLEM(ADRES(OPT(LL))))
1567  FORMAT(10X,,LOADS CARRIED :,,20I4)
 706  CONTINUE
      WRITE(6,1551)EELEN
1551  FORMAT(//,10X,,MINIMAX OPTIMAL SOLUTION VALUE IS :,,I6)
      WRITE(6,1721)JTIME
1721  FORMAT(/,10X,,TOTAL TIME IS :,,I8,///)
      CALL PORTCP(KPL,YUKLE,BOSALT,CAP,OCAP,
     & DIS,ADJ,LOAD,D,OPT,ADRES,EELEN,NPORT,CFEAS,AIR,MAIR)
      IF(CFEAS.EQ.0)STOP
      END IF
      KSET=KZ-1
      GO TO 735
 703  CONTINUE
      DELETE THE PATH
      ***************
 725  CONTINUE
      KK=OPT(KPL)
      KADR=ADRES(KK)
      PORT=AIR(KADR)
      AIRMED(PORT)=AIRMED(PORT)-1
      JTIME=JTIME-DIS(KADR)
      KPL=KPL-1
      DO 707 LL=1,KLEM(KADR)
      KLD=KLD-1
      NOD=ADJ(KADR,LL)
      CRT(NOD)=SFIRST(NOD)
      COVER(NOD)=0
 707 CONTINUE
     WRITE(6,7702)KADR,KK,KPL,KLD
7702 FORMAT(10X,'DELETE PATH :',I4,'(',I4,')  PLANES USED=',I2,
   & ' LOADS COVERED=',I3)
     WRITE(6,7710)(OPT(IKL),IKL=1,KPL)
     WRITE(6,7705)(COVER(IKL),IKL=1,NLOAD)
     KSET=KK-1
     GO TO 735
 745 CONTINUE
     RECORD THIS PATH IN MATRICES
     ***************************
3001 CONTINUE
     END IF
 700 FORMAT(2I8,20I4)
     IF(LEN.EQ.A)GO TO 996
     LLBOY=LBOY-1
     IF(LLBOY.GT.MMAX)GO TO 201
     WRITE(6,900)I,LLBOY,LEN,(PATH(J),J=1,LBOY)
 900 FORMAT(/,3X,'NO :',I5,' LBOY :',I3,' LEN :',I5,' PORT :',I2,
   & ' ==> ',20I4)
     IF(LLBOY.GT.GMAX)THEN
     GMAX=LLBOY
     WRITE(6,26000)GMAX
6000 FORMAT(/,10X,'FIRST OCURRANCE OF A',I3,' MEMBERED PATH')
     FIRST(LLBOY,1)=I
     END IF
     DO 202 J=1,LBOY
     NOD=PATH(J)
     FRE(LLBOY,NOD)=FRE(LLBOY,NOD)+1
     IF(FIRST(LLBOY,NOD).EQ.0)THEN
     FIRST(LLBOY,NOD)=I
     IF(J.GE.2)FRE(LLBOY,1)=FRE(LLBOY,1)+1
     IF(FRE(1,1).EQ.NLOAD)THEN
     IF(GECTIM.EQ.0)THEN
     GECTIM=1
     NR=MMAX
     NC=NNODE
     CALL TMAT(FRE,MMAX,HESAP,AA)
     NR=MMAX
     NC=NNODE
     CALL TMAT(FIRST,MMAX,HESAP,BB)
     LLSET=I
     END IF
     END IF
     END IF
 202 CONTINUE
     IF(I.GT.NBIN)THEN
     WRITE(6,232)
 232 FORMAT(/,10X,'***NUMBER OF SETS EXCEEDS PROGRAM LIMITS***')
     STOP
     END IF
     RECORD THIS PATH AS AN ACCUMULATED PATH
     ***************************************
     II=I
     NLEM=LLBOY
     NAIR=PATH(1)-1
     DO 217 LL=1,NLEM
     ADJ(I,LL)=PATH(LL+1)-1-NSP
 217 CONTINUE
     KLEM(II)=NLEM
     AIR(II)=NAIR
     DIS(II)=LEN
     GO TO 201
 996 CONTINUE
     WRITE(6,1550)
1550 FORMAT(///,10X,'END OF PATH LIST ACHIEVED NO FEASIBLE SOLUTION',
   & ' FOUND')
     STOP
     END
HESIS-7
```

```
C
C      **********************************
C
C
C      SEACH FOR THE BOTTLENECK OPTIMAL
C      OF THE ROUTING PROBLEM
C      BY USING ,METHOD-B, FOR BLOCKING
C      AND
C      TRY TO OBTAIN A FEASIBLE SCHEDULE
C      UNDER THIS MINIMAX OPTIMAL POINT
C
C
C      **********************************
C
C
       IMPLICIT INTEGER (A-Z)
       PARAMETER KTAPE=9
       PARAMETER PMAX=30
       PARAMETER PLMAX=30
       PARAMETER LMAX=100
       PARAMETER FMAX=100
       PARAMETER MMAX=8
       PARAMETER NBIN=1200
       PARAMETER HESAP=LMAX+PMAX+2
       DIMENSION LOAD(LMAX,2)
       DIMENSION MAIR(PMAX)
       DIMENSION PATH(FMAX)
       DIMENSION FRE(MMAX,HESAP)
       DIMENSION FIRST(MMAX,HESAP)
       DIMENSION CON1(FMAX)
       DIMENSION ADJ(NBIN,MMAX)
       DIMENSION KLEM(NBIN)
       DIMENSION AIR(NBIN)
       DIMENSION DIS(NBIN)
       DIMENSION AIRMAX(PMAX)
       DIMENSION AIRMED(PMAX)
       DIMENSION CRT(LMAX)
       DIMENSION OPT(PLMAX)
       DIMENSION COVER(LMAX)
       DIMENSION ADRES(NBIN)
       DIMENSION TOTAL(LMAX)
       DIMENSION CAP(PMAX)
       DIMENSION DCAP(PMAX)
       DIMENSION D(PMAX,PMAX)
       DIMENSION YFIRST(MMAX)
       DIMENSION SFIRST(LMAX)
C      COMMON /IMAT1/NR,NC
C      CHARACTER AA*50/,***THE FREQUENCY MATRIX***,/
C      CHARACTER BB*50/,***THE FIRST OCURRANCE MATRIX***,/
       PRINT 15000
15000  FORMAT(1H1)
       A=999099
       TFEAS=0
       GECTIM=0
       ITIME=A
       DO 1313 JK=1,PMAX
 1313  AIRMAX(JK)=0
C      GET THE INPUT DATA
C      ****************
       READ(5,*)NPORT
       DO 699 II=1,NPORT
  699  READ(5,*)(D(II,I),I=1,NPORT)
       READ(5,*)(MAIR(I),I=1,NPORT)
       READ(5,*)TLOAD
       I=0
   55  I=I+1
       READ(5,*,END=59,ERR=59)LOAD(I,1),LOAD(I,2)
       GO TO 55
   59  NLOAD=I-1
       READ(5,*)DMAX
       READ(5,*)YJKLE,BOSALT
       READ(5,*)(CAP(I),I=1,NPORT)
       READ(5,*)(DCAP(I),I=1,NPORT)
       IF(TLOAD.NE.YJKLE+BOSALT)THEN
       WRITE(6,7782)
 7782  FORMAT(//,10X,,LOADING , UNLOADING TIME INCONSISTENCY IN DATA,,
     & /,10X,,JOB TERMINATED,)
       STOP
       END IF
C      FORMULATION
C      ***********
```

```
      NAIR=0
      DO 52 I=1,NPORT
   52 NAIR=NAIR+MAIR(I)
      NR,NC=NPORT
      WRITE(6,1570)
 1570 FORMAT(///,10X,,BOTTLENECK ROUTING OF CARGO AIRCRAFT,,/,10X,
     & 36(,*,),///)
      WRITE(6,1000)NPORT
 1000 FORMAT(/,10X,,NUMBER OF PORTS      :,,I4)
C     WRITE(6,2000)NLOAD
 2000 FORMAT(/,10X,,NUMBER OF LOADS      :,,I4)
      WRITE(6,3000)TLOAD
 3000 FORMAT(/,10X,,LOADING PLUS UNLOADING TIME :,,I4)
      WRITE(6,6000)DMAX
 6000 FORMAT(/,10X,,MAXIMUM DISTANCE PERMITTED TO GENERATE PATHS :,,I4)
      WRITE(6,7783)YUKLE,BOSALT
 7783 FORMAT(/,10X,,LOADING TIME    :,I6,
     & /,10X,,UNLOADING TIME :,I6)
      WRITE(6,7784)
 7784 FORMAT(/,10X,,PORT CAPACITIES,,/,10X,15(,*,),/)
      DO 7785 I=1,NPORT
      WRITE(6,7786)I,MAIR(I),CAP(I),QCAP(I)
 7786 FORMAT(10X,,PORT :,,I2,,  PLANES :,,I2,
     & ,  SERVICE CAPACITY :,,I2,,  QUE CAPACITY :,,I2)
 7785 CONTINUE
      NPP,NPLANE=0
      DO 61 I=1,NPORT
      IF(MAIR(I).GT.0)THEN
      NPP=NPP+1
      NPLANE=NPLANE+MAIR(I)
      END IF
   61 CONTINUE
      NN=NPP+NLOAD
      NNODE=NN+2
      WRITE(6,19000)NPLANE
19000 FORMAT(/,10X,,NUMBER OF PLANES :,,I4)
      S=1
      T=NN+2
      NK=NPP+1
C     WRITE(6,7000)NNODE
C7000 FORMAT(/,10X,,TOTAL NUMBER OF NODES GENERATED :,,I4)
C     WRITE(6,8000)
C8000 FORMAT(/,10X,,MEANINGS OF NODES GENERATED,,/,10X,27(,*,),/)
      DO 1564 I=1,NPORT
 1564 CON1(I)=0
C     WRITE(6,8100)S
C8100 FORMAT(10X,,NODE,,I4,, IS THE DUMMY SOURCE NODE,)
      NOKTA=1
      DO 95 I=1,NPORT
      IF(MAIR(I).EQ.0)GO TO 95
      NOKTA=NOKTA+1
C     WRITE(6,8200)NOKTA,I
      AIRMAX(NOKTA-1)=MAIR(I)
      CON1(NOKTA-1)=I
C8200 FORMAT(10X,,NODE,,I4,, IS THE PORT,,I4)
   95 CONTINUE
      DO 1565 I=1,NPORT
 1565 MAIR(I)=CON1(I)
      DO 96 I=1,NLOAD
      NOKTA=NOKTA+1
C     WRITE(6,8300)NOKTA,LOAD(I,1),LOAD(I,2)
C8300 FORMAT(10X,,NODE,,I4,, IS THE LOAD FROM PORT,,I4,,  TO PORT,,I4)
   96 CONTINUE
C     WRITE(6,8400)T
C8400 FORMAT(10X,,NODE,,I4,, IS THE DUMMY TERMINAL NODE,)
      NR,NC=NNODE
      WRITE(6,1561)
 1561 FORMAT(/,10X,,LOADS BETWEEN PORTS,,/,10X,19(,*,),/)
      DO 1562 I=1,NLOAD
 1562 WRITE(6,1563)I,LOAD(I,1),LOAD(I,2)
 1563 FORMAT(10X,,LOAD :,,I3,,    FROM,,I4,,    TO,,I4)
C     MINIMUM CONFIGURATION SETUP
C     ***********************
C     WRITE(6,19000)NPLANE
      WRITE(6,2000)NLOAD
      N=NLOAD/NPLANE
      K=NLOAD-N*NPLANE
      PKK=K
      PK=NPLANE-K
      M1=N+1
      M2=N+2
      IF(K.EQ.0)THEN
      M1=N
      M2=N+1
      END IF
      MIK=M1*K
C     *********************************************
C     INFORMATION RETRIVIAL AND PATH SCANNING
C     *********************************************
      REWIND KTAPE
      GMAX=0
      ELEN=0
      LS=2+NPP
      LB=1+NPP+NLOAD
      KATR=NPP
      NRR=NLOAD
      DO 311 I=1,MMAX
      DO 311 J=1,NNODE
      FIRST(I,J),FRE(I,J)=0
  311 CONTINUE
      RBOY=1
      COUNT=0
C     WRITE(6,800)
C 800 FORMAT(///,10X,,THE LIST OF PATHS GENERATED,,/,10X,27(,*,),//)
      I=0
  201 I=I+1
      LEN=0
      READ(KTAPE,700,END=999,ERR=999)LBOY,LEN,(PATH(J),J=1,LBOY)
  999 CONTINUE
      IF(LEN.GT.ELEN)THEN
C     FEASIBILITY CHECKS
C     ******************
C     WRITE(6,898)LEN
C 898 FORMAT(/,10X,,FROM HERE ON PATH LENGHTS ARE,,I8,/,10X,
C    & ,START CHECKING FOR FEASIBILITY,)
      EELEN=ELEN
      ELEN=LEN
      IF(FRE(1,1).LT.NLOAD)THEN
C     WRITE(6,899)FRE(1,1),NLOAD
C 899 FORMAT(/,10X,,NUMBER OF LOADS COVERED IS,,I3,/,10X,
C    & ,WHICH IS LESS THEN TOTAL LOADS,,I3,/,10X,
C    & ,RETURN FROM FEASIBILITY CHECK,,/)
      GO TO 28001
      END IF
      IF(GMAX.LT.M1)THEN
C     WRITE(6,901)GMAX,M1
C 901 FORMAT(/,10X,,NUMBER OF LOADS PER PLANE IS,,I3,/,10X,
C    & ,WHICH IS LESS THEN,,I3,/,10X,
C    & ,RETURN FROM FEASIBILITY CHECK,,/)
      GO TO 28001
      END IF
      IF(GMAX.EQ.M1)THEN
      IF(FRE(M1,1).LT.MIK)THEN
C     WRITE(6,902)FRE(M1,1),M1,MIK
C 902 FORMAT(/,10X,,NUMBER OF DISJOINT NODES,,I4,, IN,,I3,/,10X,
C    & ,        MEMBERED PATHS WHICH IS LESS THEN,,I4,/,10X,
C    & ,RETURN FROM FEASIBILITY CHECK,,/)
      GO TO 28001
      END IF
C     CHOOSE WORST LOAD
C     *****************
      GMIN=0
      DO 341 KI=LS,LB
      DO 342 G=1,GMAX
      IF(FRE(G,KI).EQ.0)GO TO 315
  342 CONTINUE
      GG=GMAX
      GO TO 325
  315 GG=G-1
  325 IF(GG.LT.GMIN)THEN
      GMIN=GG
      END IF
  341 CONTINUE
C     WRITE(6,801)GMIN
C 801 FORMAT(/,10X,,GMIN IS,,I3)
      IF(GMAX.EQ.GMIN)GO TO 28006
      BIG=0
      DO 343 KI=LS,LB
      IF(FRE(GMIN+1,KI).GT.0)GO TO 343
      IF(FRE(GMIN,KI).LT.BIG)THEN
      BIG=FRE(GMIN,KI)
      WORST=KI
      END IF
  343 CONTINUE
```

```
      WRITE(6,812)WORST,BIG
  812 FORMAT(10X,,WORST=,,I3,,        BIG=,,I3)
      KALAN=N-GMIN
      IF(KALAN.LE.0)GO TO 28006
      MIKK=M1*(K+KALAN)
      WRITE(6,813)MIKK
  813 FORMAT(10X,,MIKK=,,I3)
      IF(FRE(M1,1).LT.MIKK)THEN
      WRITE(6,903)M1,FRE(M1,1),MIKK
  903 FORMAT(/,10X,,NUMBER OF DISJOINT NODES IN,,I3,/,10X,
     & ,              MEBERED PATHS IS,,I4,,  LESS THEN,,I4,/,10X,
     & ,RETURN FROM FEASIBILITY CHECK,)
      GO TO 28001
      END IF
      END IF
28006 CONTINUE
      FIND MINIMUM NUMBER OF PLANES NECCESSARY
      ***********************************
      TPLANE=0
      DO 444 KI=1,NNODE
  444 CON1(KI)=0
      LEFT=0
      DO 441 G=GMAX,1,-1
      LLEFT=0
      DO 442 KI=LS,LB
      IF(FRE(G,KI).EQ.0)GO TO 442
      IF(CON1(KI).EQ.1)GO TO 442
      CON1(KI)=1
      LLEFT=LLEFT+1
  442 CONTINUE
      LEFT=LEFT+LLEFT
      TT=LEFT/G
      TPLANE=TPLANE+TT
      LEFT=LEFT-TT*G
  441 CONTINUE
      WRITE(6,821)TPLANE,NPLANE
  821 FORMAT(/,10X,,AT LEAST,,I3,,  PLANE REQUIRED,,/,10X,
     & ,ALTHOUGH WE HAVE,,I3)
      IF(TPLANE.GT.NPLANE)THEN
      WRITE(6,822)
  822 FORMAT(10X,,MINIMUM PLANE CHECK FAILED,,/,10X,
     & ,RETURN FROM FEASIBILITY CHECK,)
      GO TO 28001
      END IF
C     THERE CAN BE A SOLUTION SEARCH FOR IT
C     ********************************
C     SEARCH FOR A FEASIBLE SOLUTION
C     ***************************
C     WRITE(6,7707)
C7707 FORMAT(///,10X,,SEARCHING FOR FEASIBILITY,,/,10X,25(,*,),//)
      JSET=I-1
      TCOUNT=USET
C     WRITE(6,751)LSET,JSET
C 751 FORMAT(/,10X,,LSET=,,I4,,        USET=,,I4)
C     BLOCKING OF PATHS
C     *****************
C     WRITE(6,1505)
C1505 FORMAT(/,10X,,TOTAL FREQUENCIES,,/,10X,17(,*,),/)
      DO 1501 K=I+NPP+1,NNODE-1
      KM=K-I-NPP
      TOTAL(KM)=0
      DO 1502 KK=1,GMAX
      TOTAL(KM)=TOTAL(KM)+FRE(KK,K)
 1502 CONTINUE
C     WRITE(6,1504)KM,TOTAL(KM)
C1504 FORMAT(10X,,LOAD :,,I2,,    TOTAL FREQUENCY :,,I5)
 1501 CONTINUE
      DO 1401 K=GMAX,1,-1
      DO 1402 KK=JSET,1,-1
      IF(KLEM(KK).NE.K)GO TO 1402
      ADRES(TCOUNT)=KK
      TCOUNT=TCOUNT-1
 1402 CONTINUE
 1401 CONTINUE
C     WRITE(6,1513)
C1513 FORMAT(/,10X,,THE ADRES TABLE,,/,10X,15(,*,),/)
C     DO 1514 K=1,USET
C1514 WRITE(6,1516)K,ADRES(K)
C1516 FORMAT(10X,,NEW CODE :,,I5,,       OLD CODE :,,I5)
      DO 1517 K=1,GMAX
 1517 YFIRST(K)=0
      IF(GMAX.EQ.MMAX)GO TO 1573
      DO 1530 K=GMAX+1,MMAX
1530  YFIRST(K)=JSET
1573  DO 1518 K=1,NLOAD
1518  SFIRST(K)=0
      DO 1511 K=1,USET
      KK=ADRES(K)
      KM=KLEM(KK)
      IF(YFIRST(KM).EQ.0)YFIRST(KM)=K
      IF(KM.NE.1)GO TO 1511
      IF(SFIRST(ADJ(KK,1)).EQ.0)SFIRST(ADJ(KK,1))=K
1511  CONTINUE
      WRITE(6,1526)
1526  FORMAT(/,10X,,RELATIVE FIRST OCCURRANCES OF GROUPS,,/,10X,
     & 36(,*,),/)
      DO 1521 K=1,GMAX
1521  WRITE(6,1522)K,YFIRST(K)
1522  FORMAT(10X,,GROUP SIZE :,,I2,,   RELATIVE FIRST OCCURRANCE :,,I5)
      WRITE(6,1523)
1523  FORMAT(/,10X,,RELATIVE FIRST OCCURRANCES OF LOADS,,/,10X,
     &  35(,*,),/)
      DO 1524 K=1,NLOAD
1524  WRITE(6,1525)K,SFIRST(K)
1525  FORMAT(10X,,LOAD :,,I2,,   RELATIVE FIRST OCCURRANCE :,,I5)
      DO 909 K=1,NLOAD
 909  CRT(K)=SFIRST(K)
      LSET=0
      LSET=YFIRST(M1)+PKK
      NSAYI=USET-LSET+1
      MSAYI=0
      ITIME=A
      KPL=0
      KLD=0
      DO 701 LL=1,NLOAD
 701  COVER(LL)=0
      DO 702 LL=1,NPP
 702  AIRMED(LL)=0
      KKSET=JSET
      OPTIMIZE SEARCH SCAN LIMITS
      ***************************
 735  CONTINUE
      IF(KPL.EQ.0)THEN
      JTIME=0
      MSAYI=MSAYI+1
      IF(MSAYI.GT.NSAYI)THEN
      WRITE(6,754)
 754  FORMAT(/,10X,,NO FEASIBLE SOLUTION CAN BE DETECTED,,/,10X,
     & ,CONTINUE PATH SCANNING,)
      GO TO 745
      END IF
      KSET=KKSET
      YUK=M1
      KKSET=KKSET-1
      KRT=LSET
      ELSE
      REML=NLOAD-KLD
      REMP=NPLANE-KPL
      YUK=REML/REMP
      IF(YUK*REMP.NE.REML)YUK=YUK+1
      KRT=YFIRST(YUK)
      LRT=0
      DO 904 K=1,NLOAD
      IF(CRT(K).GT.LRT)LRT=CRT(K)
 904  CONTINUE
      END IF
      IF(LRT.GT.KRT)KRT=LRT
      WRITE(6,7706)KSET,KRT,YUK
7706  FORMAT(10X,,THIS COVERAGE RESULTED WITH     ,,
     & ,KSET=,,I4,,       KRT=,,I4,,       YUK=,,I4,
     & /,10X,50(,-,))
      IF(KSET.GE.KRT)THEN
      GO TO 715
      ELSE
      GO TO 725
      END IF
      SEARCH FOR A FEASIBLE PATH
      **************************
 715  CONTINUE
      DO 703 K=KSET,KRT,-1
      KADR=ADRES(K)
      PORT=AIR(KADR)
      IF(AIRMED(PORT).EQ.AIRMAX(PORT))GO TO 703
      IF(JTIME+DIS(KADR).GE.ITIME)GO TO 703
      DO 704 LL=1,KLEM(KADR)
      NOD=ADJ(KADR,LL)
      IF(COVER(NOD).EQ.1)GO TO 703
 704  CONTINUE
      INCLUDE THE PATH
      ***************
      KPL=KPL+1
      OPT(KPL)=K
      KZ=0
      AIRMED(PORT)=AIRMED(PORT)+1
      JTIME=JTIME+DIS(KADR)
      DO 705 LL=1,KLEM(KADR)
      KLD=KLD+1
      NOD=ADJ(KADR,LL)
      CRT(NOD)=0
      COVER(NOD)=1
 705  CONTINUE
      WRITE(6,7701)KADR,K,KPL,KLD
7701  FORMAT(10X,'INCLUDE PATH :',I4,'(',I4,')  PLANES USED=',I2,
     & '  LOADS COVERED=',I3)
      WRITE(6,7710)(OPT(IKL),IKL=1,KPL)
7710  FORMAT(10X,'PLANE ASSIGNMENTS:',10(I4,(',')))
      WRITE(6,7705)(COVER(IKL),IKL=1,NLOAD)
7705  FORMAT(10X,'LOAD COVERAGE IS AS FOLLOWS : ',
     &       20(I3))
      IF(KLD.EQ.NLOAD)THEN
      WRITE(6,752)
 752  FORMAT(//////,10X,'MINIMAX OPTIMAL SOLUTION FOUND AND GIVEN BELOW',
     & /,10X,46('*'),/)
      IFEAS=IFEAS+1
      ITIME=JTIME
      WRITE(6,1597)IFEAS
1597  FORMAT(/,10X,'ALTERNATE SOLUTION NUMBER :',I3)
      DO 706 LL=1,KPL
      WRITE(6,753)LL,OPT(LL),ADRES(OPT(LL)),MAIR(AIR(ADRES(OPT(LL)))),
     & DIS(ADRES(OPT(LL)))
 753  FORMAT(/,10X,'PLANE NO :',I2,'       TO  RELATIVE PATH :',I6,
     & '    TO PATH :',I6,'    PORT :',I2,'      TIME :',I6)
      WRITE(6,1567)(ADJ(ADRES(OPT(LL)),JK),JK=1,KLEM(ADRES(OPT(LL))))
1567  FORMAT(10X,'LOADS CARRIED :',20I4)
 706  CONTINUE
      WRITE(6,1551)EELEN
1551  FORMAT(///,10X,'MINIMAX OPTIMAL SOLUTION VALUE IS :',I6)
      WRITE(6,1721)JTIME
1721  FORMAT(/,10X,'TOTAL TIME IS :',I8,///)
      CALL PORTCP(KPL,YUKLE,BOSALT,CAP,QCAP,
     & DIS,ADJ,LOAD,D,OPT,ADRES,EELEN,NPORT,CFEAS,AIR,MAIR)
      IF(CFEAS.EQ.0)STOP
      END IF
      KSET=K-1
      GO TO 735
 703  CONTINUE
      DELETE THE PATH
      **************
 725  CONTINUE
      KK=OPT(KPL)
      KADR=ADRES(KK)
      PORT=AIR(KADR)
      AIRMED(PORT)=AIRMED(PORT)-1
      JTIME=JTIME-DIS(KADR)
      KPL=KPL-1
      DO 707 LL=1,KLEM(KADR)
      KLD=KLD-1
      NOD=ADJ(KADR,LL)
      CRT(NOD)=SFIRST(NOD)
      COVER(NOD)=0
 707  CONTINUE
      WRITE(6,7702)KADR,KK,KPL,KLD
C7702 FORMAT(10X,'DELETE PATH :',I4,'(',I4,')  PLANES USED=',I2,
C    & '  LOADS COVERED=',I3)
C     WRITE(6,7710)(OPT(IKL),IKL=1,KPL)
C     WRITE(6,7705)(COVER(IKL),IKL=1,NLOAD)
      KSET=KK-1
      GO TO 735
 745  CONTINUE
C     RECORD THIS PATH IN MATRICES
C     ****************************
28001 CONTINUE
      END IF
 700  FORMAT(2I8,20I4)
      IF(LEN.EQ.A)GO TO 996
      LLBOY=LBOY-1
      IF(LLBOY.GT.MMAX)GO TO 201
      WRITE(6,900)I,LLBOY,LEN,(PATH(J),J=1,LBOY)
 900  FORMAT(/,3X,'NO :',I5,' LBOY :',I3,' LEN :',I5,' PORT :',I2,
     & '  ===> ',20I4)
      IF(LLBOY.GT.GMAX)THEN
      GMAX=LLBOY
      WRITE(6,26000)GMAX
6000  FORMAT(/,10X,'FIRST OCURRANCE OF A',I3,' MEMBERED PATH')
      FIRST(LLBOY,1)=I
      END IF
      DO 202 J=1,LBOY
      NOD=PATH(J)
      FRE(LLBOY,NOD)=FRE(LLBOY,NOD)+1
      IF(FIRST(LLBOY,NOD).EQ.0)THEN
      FIRST(LLBOY,NOD)=I
      IF(J.GE.2)FRE(LLBOY,1)=FRE(LLBOY,1)+1
      IF(FRE(1,1).EQ.NLOAD)THEN
      IF(GECTIM.EQ.0)THEN
      GECTIM=1
      NR=MMAX
      NC=NNODE
      CALL TMAT(FRE,MMAX,HESAP,AA)
      NR=MMAX
      NC=NNODE
      CALL TMAT(FIRST,MMAX,HESAP,BB)
      LSET=I
      END IF
      END IF
      END IF
 202  CONTINUE
      IF(I.GT.NBIN)THEN
      WRITE(6,232)
 232  FORMAT(/,10X,'***NUMBER OF SETS EXCEEDS PROGRAM LIMITS***')
      STOP
      END IF
      RECORD THIS PATH AS AN ACCUMULATED PATH
      ****************************************
      II=I
      NLEM=LLBOY
      NAIR=PATH(1)-1
      DO 217 LL=1,NLEM
      ADJ(I,LL)=PATH(LL+1)-1-NPP
 217  CONTINUE
      KLEM(II)=NLEM
      AIR(II)=NAIR
      DIS(II)=LEN
      GO TO 201
 995  CONTINUE
      WRITE(6,1550)
1550  FORMAT(///,10X,'END OF PATH LIST ACHIEVED NO FEASIBLE SOLUTION',
     & ' FOUND')
      STOP
      END
THESIS-8
```

```
      SUBROUTINE PORTCP(KPL,YUKLE,BOSALT,CAP,QCAP,
     & DIS,ADJ,LOAD,D,OPT,ADRES,EELEN,NPORT,CFEAS,AIR,MAIR)

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

      SCHEDULE PLANES SUCH THAT PORT CAPACITIES
      DO NOT VIOLATED
      CHECK AIRPORT CAPACITY CONSTRAINTS

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

      CAP(I)    = CAPACITY OF PORT I

      USE(I)    = CURRENT USAGE OF PORT I

      QCAP(I)   = QUE CAPACITY OF PORT I

      QUSE(I)   = CURRENT QUE USAGE OF PORT I

      SLACK(J)  = CURRENT SLACK OF PLANE J

      COMPT(J)  = COMPLETION TIME OF CURRENT STATUS OF PLANE J

      POINT(J)  = CURRENT JOB POINTER OF PLANE J

      STATUS(J) = CURRENT STATUS OF PLANE J
                = 1 , EMPTY FLIGHT
                = 2 , FUL  FLIGHT
                = 3 , LOADING
                = 4 , UNLOADIG
                = 5 , WAITING FOR LOADING
                = 6 , WAITING FOR UNLOADING
                = 7 , FINISHED JOB

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

      IMPLICIT INTEGER (A-Z)
      PARAMETER PMAX=30
      PARAMETER PLMAX=30
      PARAMETER MMAX=30
      PARAMETER LMAX=100
      PARAMETER NBIN=1200
      DIMENSION CAP(PMAX)
      DIMENSION USE(PMAX)
      DIMENSION QCAP(PMAX)
      DIMENSION QUSE(PMAX)
      DIMENSION MAIR(PMAX)
      DIMENSION DIS(NBIN)
      DIMENSION ADJ(NBIN,MMAX)
      DIMENSION ADRES(NBIN)
      DIMENSION AIR(NBIN)
      DIMENSION D(PMAX,PMAX)
      DIMENSION LOAD(LMAX,2)
      DIMENSION RUT(PLMAX)
      DIMENSION OPT(PLMAX)
      DIMENSION QUE(PMAX,PLMAX)
      DIMENSION SLACK(PLMAX)
      DIMENSION COMPT(PLMAX)
      DIMENSION STATUS(PL MAX)
      DIMENSION POINT(PLMAX)
      A=999999
      CFEAS=0
      DO 2511 K=1,KPL
 2511 RUT(K)=ADRES(OPT(K))
      DUE=EELEN
      DO 2512 K=1,KPL
 2512 SLACK(K)=DUE-DIS(RUT(K))
      CLOCK=0
      DO 2513 K=1,NPORT
 2513 USE(K),QUSE(K)=0
      INITIALIZE PLANE STATUS
      ***********************
      DO 2531 K=1,KPL
      POINT(K)=1
      ORIG=MAIR(AIR(RUT(K)))
      DES=LOAD(ADJ(RUT(K),1),1)
      IF(ORIG.EQ.DES)THEN
      QUSE(ORIG)=QUSE(ORIG)+1
      QUE(ORIG,QUSE(ORIG))=K
      STATUS(K)=5
      COMPT(K)=A
      ELSE
      TT=D(ORIG,DES)
      STATUS(K)=1
      COMPT(K)=TT
      WRITE(6,2552)CLOCK,K,ORIG,DES
      END IF
 2531 CONTINUE
      GO TO 2575
      UPDATE CLOCK
      ************
 2515 CONTINUE
      DO 2526 K=1,KPL
      IF(STATUS(K).EQ.7)GO TO 2526
      GO TO 2565
 2526 CONTINUE
      WRITE(6,2558)
 2558 FORMAT(//,10X,,SOLUTION FEASIBLE JOB COMPLETED,,////)
      CFEAS=0
      RETURN
 2565 CONTINUE
      MINA=A
      OCLOCK=CLOCK
      DO 2501 K=1,KPL
      IF(COMPT(K).LT.MINA)MINA=COMPT(K)
 2501 CONTINUE
      CLOCK=MINA
      UPDATE SLACKS OF PLANES
      ***********************
      WAIT=CLOCK-OCLOCK
      DO 2522 K=1,NPORT
      IF(QUSE(K).EQ.0)GO TO 2522
      DO 2523 KK=1,QUSE(K)
      LPL=QUE(K,KK)
      SLACK(LPL)=SLACK(LPL)-WAIT
      IF(SLACK(LPL).LT.0)THEN
      WRITE(6,2557)LPL,K
 2557 FORMAT(10X,,PLANE ,,I2,, CANNOT FINISH JOB AT THIS MINIMAX VALUE,,
     & /,10X,,BECAUSE OF WAITING AT THE QUE OF PORT ,,I2,
     &  /,10X,,RETURN FROM FEASIBILITY CHECK,)
      CFEAS=1
      RETURN
      END IF
 2523 CONTINUE
 2522 CONTINUE
C     UPDATE PLANE STATUS
C     *******************
      DO 2502 K=1,KPL
      IF(COMPT(K).NE.CLOCK)GO TO 2502
      POS=POINT(K)
      LPOS=ADJ(RUT(K),POS)
C     IF CLOCK = END OF FULL FLIGHT
C     *****************************
      IF(STATUS(K).EQ.2)THEN
      DES=LOAD(LPOS,2)
      QUSE(DES)=QUSE(DES)+1
      QUE(DES,QUSE(DES))=K
      COMPT(K)=A
      STATUS(K)=6
      GO TO 2502
      END IF
C     IF CLOCK = END OF LOADING
C     *************************
      IF(STATUS(K).EQ.3)THEN
      ORIG=LOAD(LPOS,1)
      USE(ORIG)=USE(ORIG)-1
      DES=LOAD(LPOS,2)
      TT=D(ORIG,DES)
      COMPT(K)=CLOCK+TT
      STATUS(K)=2
      WRITE(6,2551)CLOCK,K,LPOS,ORIG,DES
 2551 FORMAT(10X,,TIME :,,I5,,    PLANE :,,I2,
     & ,  START CARRYING LOAD ,,I2,,  FROM ,,I2,,  TO ,,I2)
      GO TO 2502
      END IF
C     IF CLOCK = END OF UNLOADING
C     ***************************
      IF(STATUS(K).EQ.4)THEN
      ORIG=LOAD(ADJ(RUT(K),POS),2)
      USE(ORIG)=USE(ORIG)-1
      IF(POS.EQ.MMAX)GO TO 2505
      LLPOS=ADJ(RJT(K),POS+1)
      IF(LLPOS.EQ.0)GO TO 2505
      DES=LOAD(ADJ(RJT(K),POS+1),1)
      POINT(K)=POS+1
      IF(ORIG.EQ.DES)THEN
      QUSE(DES)=QUSE(DES)+1
      QUE(DES,QUSE(DES))=K
      COMPT(K)=A
      STATUS(K)=5
      GO TO 2502
      ELSE
      TT=D(ORIG,DES)
      COMPT(K)=CLOCK+TT
      STATUS(K)=1
      WRITE(6,2552)CLOCK,K,ORIG,DES
 2552 FORMAT(10X,'TIME :',I5,'   PLANE :',I2,
     & '  START FLYING EMPTY FROM ',I2,'  TO ',I2)
      GO TO 2502
      END IF
 2505 CONTINUE
      STATUS(K)=7
      COMPT(K)=A
      WRITE(6,2555)CLOCK,K
 2555 FORMAT(10X,'TIME :',I5,'   PLANE :',I2,'  COMPLETED JOB')
      GO TO 2502
      END IF
      IF CLOCK = END OF EMPTY FLIGHT
      ****************************
      IF(STATUS(K).EQ.1)THEN
      DES=LOAD(LPOS,1)
      QUSE(DES)=QUSE(DES)+1
      QUE(DES,QUSE(DES))=K
      COMPT(K)=A
      STATUS(K)=5
      GO TO 2502
      END IF
 2502 CONTINUE
      DECIDE ON PRIORITES
      ******************
 2575 CONTINUE
      DO 2506 K=1,NPORT
      IF(QUSE(K).EQ.0)GO TO 2506
      IF(CAP(K)-USE(K).GE.QUSE(K))THEN
      DO 2507 KK=1,QUSE(K)
      LPL=QUE(K,KK)
      QUE(K,KK)=0
      USE(K)=USE(K)+1
      QUSE(K)=QUSE(K)-1
      IF(STATUS(LPL).EQ.5)THEN
      TT=YUKLE
      STATUS(LPL)=3
      WRITE(6,2553)CLOCK,LPL,K
 2553 FORMAT(10X,'TIME :',I5,'   PLANE :',I2,
     & '  START LOADING AT PORT ',I2)
      ELSE
      TT=BOSALT
      STATUS(LPL)=4
      WRITE(6,2554)CLOCK,LPL,K
 2554 FORMAT(10X,'TIME :',I5,'   PLANE :',I2,
     & '  START UNLOADING AT PORT ',I2)
      END IF
      COMPT(LPL)=CLOCK+TT
 2507 CONTINUE
      GO TO 2506
      ELSE
 2535 CONTINUE
      IF(USE(K).EQ.CAP(K))GO TO 2506
      MINA=A
      DO 2508 KK=1,QUSE(K)
      LPL=QUE(K,KK)
      IF(SLACK(LPL).LT.MINA)THEN
      MINA=SLACK(LPL)
      CK=KK
      END IF
 2508 CONTINUE
      LPL=QUE(K,CK)
      QUE(K,CK)=0
      IF(CK.EQ.QUSE(K))GO TO 2525
      DO 2509 KK=CK+1,QUSE(K)
 2509 QUE(K,KK-1)=QUE(K,KK)
 2525 CONTINUE
      QUSE(K)=QUSE(K)-1
      USE(K)=USE(K)+1
      IF(STATUS(LPL).EQ.5)THEN
      TT=YUKLE
      STATUS(LPL)=3
      WRITE(6,2553)CLOCK,LPL,K
      ELSE
      TT=BOSALT
      STATUS(LPL)=4
      WRITE(6,2554)CLOCK,LPL,K
      END IF
      COMPT(LPL)=CLOCK+TT
      GO TO 2535
      END IF
2506  CONTINUE
      WRITE(6,2571)CLOCK
2571  FORMAT(10X,'QUEUES AT THE PORTS AT TIME :',I6)
      TM=0
      DO 2572 K=1,NPORT
      IF(QUSE(K).EQ.0)GO TO 2572
      TM=1
      WRITE(6,2573)K,(QUE(K,J),J=1,QUSE(K))
2573  FORMAT(10X,'PORT :',I2,' : ',20(I2,','))
2572  CONTINUE
      IF(TM.EQ.0)WRITE(6,2574)
2574  FORMAT(10X,'NO QUE IS PRESENT')
      WRITE(6,2577)
2577  FORMAT(//)
      DO 2521 K=1,NPORT
      IF(QUSE(K).GT.QCAP(K))THEN
      CFEAS=1
      WRITE(6,2556)K
2556  FORMAT(10X,'QUE CAPACITY OF PORT ',I2,' IS VIOLATED',
     & /,10X,'RETURN FROM FEASIBILITY CHECK')
      RETURN
      END IF
2521  CONTINUE
      GO TO 2515
      END
HESIS-9
```

```
      SUBROUTINE TIMER(B)
      *******************************************************
      THIS SUBROUTINE PRINTS OUT THE GIVEN MESSAGE
      WITH THE TIME TAKEN FROM THE SYSTEM CONSOLE
      *******************************************************
      CHARACTER B*50,I*6,J*6
      CALL ADATE(I,J)
      WRITE(6,1)J,B
    1 FORMAT(1X,'CURRENT TIME :',1X,A6,1X,A50)
      RETURN
      END
HESIS-10
```

```
      SUBROUTINE IMAT(IA,N5,N6,B)
C     ***************************************************************
C     THIS SUBROUTINE OUTPUTS AN INTEGER MATRIX OF ANY SIZE
C     ***************************************************************
      DIMENSION IA(N5,N6)
      CHARACTER A*3,B*50
      COMMON /IMAT1/N1,N2
      NR=N5
      NC=N6
      IF(N2.EQ.0)GO TO 8
      NC=MIN(NC,N2)
      IF(N1.EQ.0)GO TO 8
      NR=MIN(NR,N1)
    8 CONTINUE
      WRITE(5,9)B
      A=' **'
      N=0
      K=NC/40
      J=NC-K*40
      L=K+1
      IF(J.EQ.0)L=L-1
      IF(L.LE.1)GO TO 5
      N=1
      GO TO 6
    5 WRITE(5,11)(II,II=1,NC)
      WRITE(5,13)(A,II=1,NC)
    6 DO 1 I=1,NR
      NC1=1
      DO 2 LL=1,L
      NC2=NC1+MIN(39,NC-NC1)
      IF(N.EQ.0)GO TO 720
      WRITE(5,11)(II,II=NC1,NC2)
  720 WRITE(5,12)I,(IA(I,II),II=NC1,NC2)
    2 NC1=NC2+1
    1 CONTINUE
    9 FORMAT(///,8X,A50)
   10 FORMAT(1X,'ROW',I3)
   11 FORMAT(4X,'COL',1X,40I3)
   13 FORMAT(8X,40A3)
   12 FORMAT(1X,'ROW',I3,'*',40I3)
      N1,N2=0
      RETURN
      END
THESIS-11
```

```
      SUBROUTINE PATOUT(IPATH,L,N)
C     ************************
C     SUBROUTINE PATH OUTPUTTER
C     ************************
      DIMENSION IPATH(N)
      WRITE(6,1)(IPATH(I),I=1,L)
    1 FORMAT((5X,30(I3,',')))
      RETURN
      END
,THESIS-12
```

```
      BOTTLENECK ROUTING OF MILITARY CARGO AIRCRAFT
      BY USING SET PARTITIONING ALGORITHM

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

      IMPLICIT INTEGER (A-Z)
      PARAMETER KTAPE=9
      PARAMETER PMAX=30
      PARAMETER PLMAX=30
      PARAMETER LMAX=100
      PARAMETER FMAX=100
      PARAMETER MMAX=8
      PARAMETER NBIN=1000
      PARAMETER NOTJZ=25
      PARAMETER NELLI=250
      PARAMETER NON=PMAX
      PARAMETER HESAP=LMAX+PMAX+2
      DIMENSION LOAD(LMAX,2)
      DIMENSION MAIR(PMAX)
      DIMENSION PATH(FMAX)
      DIMENSION ADJ(NOTJZ,NBIN)
      DIMENSION KLEM(NBIN)
      DIMENSION AIR(NBIN)
      DIMENSION AIRMAX(NON)
      DIMENSION AIRMED(NON)
      DIMENSION MARK(NBIN)
      DIMENSION DIS(NBIN)
      DIMENSION NEB(NBIN)
      DIMENSION E(NOTJZ)
      DIMENSION B(NOTJZ,2)
      DIMENSION BBAR(NOTJZ,2)
      DIMENSION BLOCK(NOTJZ,NELLI)
      DIMENSION IBOPT(NOTJZ)
C     COMMON /IMAT1/NR,NC
C     CHARACTER AA*50/,***THE FREQUENCY MATRIX***,/
C     CHARACTER BB*50/,***THE FIRST OCURRANCE MATRIX***,/
      PRINT 15000
15000 FORMAT(1H1)
      A=999999
      SECTIM=0
      DO 1313 JK=1,PMAX
 1313 AIRMAX(JK)=0
C     GET THE INPUT DATA
C     ******************
      READ(5,*)NPORT
      DO 699 II=1,NPORT
  699 READ(5,*)(MAIR(I),I=1,NPORT)
      READ(5,*)(MAIR(I),I=1,NPORT)
      READ(5,*)TLOAD
      I=0
   55 I=I+1
      READ(5,*,END=59,ERR=59)LOAD(I,1),LOAD(I,2)
      GO TO 55
   59 NLOAD=I-1
      READ(5,*)DMAX
      READ(5,*)PATLEN,UPPER
      IPRINT=1
C     FORMULATION
C     ***********
      NAIR=0
      DO 52 I=1,NPORT
   52 NAIR=NAIR+MAIR(I)
      NR,NC=NPORT
      WRITE(6,1570)
 1570 FORMAT(///,10X,,BOTTLENECK ROUTING OF CARGO AIRCRAFT,,/,10X,
     & 36(,*,),///)
      WRITE(6,1000)NPORT
 1000 FORMAT(/,10X,,NUMBER OF PORTS          :,,I4)
C     WRITE(6,2000)NLOAD
 2000 FORMAT(/,10X,,NUMBER OF LOADS          :,,I4)
      WRITE(6,3000)TLOAD
 3000 FORMAT(/,10X,,LOADING PLUS UNLOADING TIME :,,I4)
      WRITE(6,6000)DMAX
 6000 FORMAT(/,10X,,MAXIMUM DISTANCE PERMITTED TO GENERATE PATHS :,,I4)
      WRITE(6,6791)PATLEN
 6791 FORMAT(/,10X,,MAXIMUM PATH LENGHT FOR SPP OPTIMIZATION :,,I6)
      WRITE(6,6792)UPPER
```

```
6792  FORMAT(/,10X,,UPPER BOUND ON THE MINIMUM TOTAL COST :,,I8)
      WRITE(6,4000)
 4000 FORMAT(/,10X,,AVAILABLE PLANES AT AIRPORTS,,/,10X,28(,*,))
      DO 54 I=1,NPORT
      WRITE(6,5000)I,MAIR(I)
 5000 FORMAT(10X,,PORT,,I4,10X,,AVAILABLE PLANES,,I5)
   54 CONTINUE
      NPP,NPLANE=0
      DO 61 I=1,NPORT
      IF(MATR(I).GT.0)THEN
      NPP=NPP+1
      NPLANE=NPLANE+MATR(I)
      END IF
   61 CONTINUE
      NN=NPP+NLOAD
      NNODE=NN+2
      WRITE(6,19000)NPLANE
19000 FORMAT(/,10X,,NUMBER OF PLANES :,,I4)
      S=1
      T=NN+2
      NK=NPP+1
C     WRITE(6,7000)NNODE
C7000 FORMAT(/,10X,,TOTAL NUMBER OF NODES GENERATED :,,I4)
C     WRITE(6,8000)
C8000 FORMAT(/,10X,,MEANINGS OF NODES GENERATED,,/,10X,27(,*,),/)
C     WRITE(6,8100)S
C8100 FORMAT(10X,,NODE,,I4,, IS THE DUMMY SOURCE NODE,)
      NOKTA=1
      DO 95 I=1,NPORT
      IF(MATR(I).EQ.0)GO TO 95
      NOKTA=NOKTA+1
C     WRITE(6,8200)NOKTA,I
      AIRMAX(NOKTA-1)=MAIR(I)
C8200 FORMAT(10X,,NODE,,I4,, IS THE PORT,,I4)
   95 CONTINUE
      DO 96 I=1,NLOAD
      NOKTA=NOKTA+1
C     WRITE(6,8300)NOKTA,LOAD(I,1),LOAD(I,2)
C8300 FORMAT(10X,,NODE,,I4,, IS THE LOAD FROM PORT,,I4,, TO PORT,,I4)
   96 CONTINUE
C     WRITE(6,8400)T
C8400 FORMAT(10X,,NODE,,I4,, IS THE DUMMY TERMINAL NODE,)
      WRITE(6,1561)
 1561 FORMAT(/,10X,,LOADS BETWEEN PORTS,,/,10X,19(,*,),/)
      DO 1562 I=1,NLOAD
 1562 WRITE(6,1563)I,LOAD(I,1),LOAD(I,2)
 1563 FORMAT(10X,,LOAD :,,I3,,   FROM,,I4,,   TO,,I4)
      WRITE(6,2000)NLOAD
C     ************************************
C     INFORMATION RETRIVIAL AND PATH SCANNING
C     ************************************
      REWIND KTAPE
      GMAX=0
      ELEN=0
      LS=2+NPP
      LB=1+NPP+NLOAD
      KAIR=NPP
      NRR=NLOAD
      RBOY=1
      COUNT=0
C     WRITE(6,800)
C 800 FORMAT(///,10X,,THE LIST OF PATHS GENERATED,,/,10X,27(,*,),//)
      I=0
  201 I=I+1
      LEN=A
      READ(KTAPE,700,END=999,ERR=999)LBOY,LEN,(PATH(J),J=1,LBOY)
  700 FORMAT(2I8,20I4)
  999 CONTINUE
      IF(LEN.GT.PATLEN)THEN
C     CALL SPP TO OPTIMIZE
C     ********************
      NRR=NLOAD
      NP=I-1
      CALL SPP1(NRR,KAIR,AIRMAX,KLEM,AIR,DIS,ADJ,NP,MARK,
     & FEAS,IPRINT,E,B,BBAR,BLOCK,NEB,AIRMED,IBOPT,NELLI,
     & NBIN,NOTUZ,NON,UPPER)
      STOP
      END IF
C     RECORD THIS PATH AS AN ACCUMULATED PATH
C     **********************************
      II=I
      LLBOY=LBOY-1
      NLEM=LLBOY
      DO 1551 KXI=2,LBOY
1551  E(KXI-1)=PATH(KXI)-1-NPP
      DO 206 KX=1,NLEM
      MINA=A
      DO 207 J=KX,NLEM
      IF(E(J).LT.MINA)THEN
      MINA=E(J)
      JC=J
      END IF
 207  CONTINUE
      IDUM=E(JC)
      E(JC)=E(KX)
      E(KX)=IDUM
 206  CONTINUE
      MARK(II)=E(1)
      DO 203 J=1,NLEM
 203  ADJ(E(J),II)=1
      DIS(II)=LEN
      NAIR=PATH(1)-1
      KLEM(II)=NLEM
      AIR(II)=NAIR
      GO TO 201
 995  CONTINUE
      WRITE(6,1550)
1550  FORMAT(///,10X,'END OF PATH LIST ACHIEVED NO FEASIBLE SOLUTION',
     & ' FOUND')
      STOP
      END
HESIS-13
```

```
      SUBROUTINE SPP1(NR,MAIR,AIRMAX,LEM,AIR,C,ADJ,NP,MARK,FEAS,
     & IPRINT,E,B,BBAR,BLOCK,NEB,AIRMED,IBOPT,
     & NELLI,NBIN,NOTUZ,NON,UPPER)
C
C     ******************************************************************
C     THIS SUBROUTINE SOLVES SET PARTITIONING PROBLEMS OF EITHER
C     CONSTRAINED OR UNCONSTRAINED TYPE ,
C     OBJECTIVE FUNCTION CAN BE EITHER MINIMIZE TOTAL COST OR
C     MINIMIZE MAXIMUM COST
C     ******************************************************************
C
C
C     ******************************************************************
C
C     DEFINITIONS :
C
C     IOBJ   = OBJECTIVE FUNCTION CONTROL VARIABLE
C            = 1 , MINIMIZE TOTAL COST
C            = 2 , MINIMIZE MAXIMUM COST
C     ICONS  = CONSTRAINT TYPE CONTROL VARIABLE
C            = 1 , PROBLEM IS UNCONSTRAINED
C            = 2 , PROBLEM IS CONSTRAINED
C     IPRINT = PRINTOUT OPTION
C            = 2 , DETAILED PRINTOUT IS PRODUCED INCLUDING
C                  FEASIBLE SOLUTIONS AND MATRICES
C            = 1 , MEDIUM PRINTOUT IS PRODUCED INCLUDING
C                  FEASIBLE SOLUTIONS ONLY
C            = 0 , SHORTEST POSSIBLE PRINTOUT WILL BE PRODUCED
C
C     ******************************************************************
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION      E(NOTUZ),ADJ(NOTUZ,NBIN),MARK(NBIN),
     & C(NBIN),B(NOTUZ,2),BBAR(NOTUZ,2),BLOCK(NOTUZ,NELLI),
     & NEB(NBIN),LEM(NBIN),AIR(NBIN),
     & AIRMED(NON),AIRMAX(NON),IBOPT(NOTUZ)
      COMMON /IMAT1/N1,N2
C     CHARACTER LAF*50/,THE NODE-SET ADJACENCY MATRIX,/
C     CHARACTER LAF2*50/,BLOCK ADRESSES,/
C     CHARACTER AA*50/,SPP HAS STARTED,/
C     CHARACTER BB*50/,SPP HAS FINNED,/
C     CALL TIMER(AA)
C     *******************************
C     INITIALIZE AND GET THE INPUT DATA
C     *******************************
      A=UPPER
      ABC=A-1
      NV=NOTUZ
      ITFEAS=0
C     **********************
C     NR = NUMBER OF NODES (ROWS)
C     NP = NUMBER OF PATHS (SETS)
C     **********************
      IOBJ=1
      ICONS=2
      FEAS=1
   98 CONTINUE
      DO 4 I=1,NP
      IBL=MARK(I)
      NEB(IBL)=NEB(IBL)+1
      IF(NEB(IBL).GT.NELLI)GO TO 1385
      BLOCK(IBL,NEB(IBL))=I
      MARK(I)=0
    4 CONTINUE
      MAXA=0
      DO 131 I=1,NR
      IF(NEB(I).GT.MAXA)MAXA=NEB(I)
  131 CONTINUE
      LBOY=0
      IF(ICONS.EQ.1)THEN
      DO 137 I=1,NP
      AIR(I)=1
  137 CONTINUE
      END IF
      N1=NR
      N2=NP
      N100=NBIN
      WRITE(6,233)NP
      WRITE(6,234)MAXA
  233 FORMAT(//,10X,,NUMBER OF PATHS GIVEN TO SPP :,,I6)
```

```
      IF(IPRINT.EQ.2)THEN
      CALL IMAT(ADJ,NOTUZ,NBIN,LAF)
      N1=NR
      N2=MAXA
 234  FORMAT(/,10X,,MAXIMUM BLOCK SIZE,,I5)
      CALL IMAT(BLOCK,NOTUZ,NBIN,LAF2)
      END IF
      IF(IPRINT.GE.1)THEN
      WRITE(6,510)
 510  FORMAT(//,10X,,***SET INFORMATION***,,//)
C     DO 21 I=1,NP
      WRITE(6,520)I,C(I),LEM(I),AIR(I)
 520  FORMAT(2X,,SET,,I5,2X,,COST,,I8,2X,
     & ,NO OF ELEM.,,I3,2X,,SUPPLY POINT,,I2)
  21  CONTINUE
      END IF
      MAXMIN=A
      IF(ICONS.EQ.2)THEN
      WRITE(6,1040)
1040  FORMAT(//,10X,,***RESOURCE AVAILIBILITIES AT SUPPLY POINTS***,,//)
      DO 22 I=1,MAIR
      WRITE(6,1041)I,AIRMAX(I)
  22  CONTINUE
1041  FORMAT(I2X,,SUPPLY POINT.,I5,5X,,AVAILABLE RESOURCE,,I5)
C     END IF
      N1=NR
      N2=MAXA
C     ******
C     STEP 1
C     ******
 101  CONTINUE
      LBOY=0
      DO 787 I=1,NP
 787  MARK(I)=0
      DO 788 I=1,NON
 788  AIRMED(I)=0
      DO 5 I=1,NV
      B(I,1),B(I,2)=0
   5  CONTINUE
      DO 6 I=1,NR
      E(I)=0
   6  CONTINUE
      Z=0
      ZBAR=UPPER
C     WRITE(6,1002)
1002  FORMAT(/,10X,,STEP 1 IS COMPLETED,)
C     ******
C     STEP 2
C     ******
 102  CONTINUE
      DO 7 I=1,NR
      IF(E(I).EQ.0)GO TO 45
   7  CONTINUE
  45  CONTINUE
      P=I
      IF(NEB(P).EQ.0)GO TO 104
      DO 8 J=1,NEB(P)
      ISET=BLOCK(P,J)
      IF(MARK(ISET).EQ.1)GO TO 8
      ICON=2
      MARK(ISET)=1
      LMARK=ISET
C     WRITE(6,1003)P,ISET
1003  FORMAT(/,10X,,AT STEP 2 : CHOSEN BLOCK P IS,,I4,/,
     & 22X,,LOWEST COST REAL SET IS,,I4)
C     WRITE(6,1012)ISET
1012  FORMAT(10X,,MARK REAL SET,,I4)
      GO TO 55
   8  CONTINUE
  55  CONTINUE
C     ******
C     STEP 3
C     ******
 103  CONTINUE
      I=1
      IF(NEB(P).EQ.1)GO TO 65
      IF(NEB(P).EQ.0)GO TO 104
      DO 9 I=NEB(P),1,-1
      KI=BLOCK(P,I)
      IF(MARK(KI).EQ.1)GO TO 65
   9  CONTINUE
  65  CONTINUE
```

```
      IBEG=I
      WRITE(6,1004)IBEG
 1004 FORMAT(/,10X,,AT STEP 3 : BEGIN WITH REL SET,,I4)
      DO 10 J=IBEG,NFB(P)
      ISET=BLOCK(P,J)
      WRITE(6,1008)ISET
 1008 FORMAT(15X,,TRY REAL SET,,I4)
      IF(Z+C(ISET).GE.ZBAR)GO TO 104
      IF(ICONS.EQ.2)THEN
      IF(AIRMED(AIR(ISET)).GE.AIRMAX(AIR(ISET)))GO TO 10
      END IF
      DO 11 I=1,NR
      IF(E(I).EQ.0)GO TO 11
      IF(ADJ(I,ISET).EQ.1)GO TO 10
   11 CONTINUE
C     WRITE(6,1013)LMARK
      MARK(LMARK)=0
      WRITE(6,1005)ISET
 1005 FORMAT(10X,,FOR REAL SET,,I4,2X,,CONDITION 3-I SATISFIED,)
      LMARK=ISET
      MARK(ISET)=1
C     WRITE(6,1012)ISET
      GO TO 105
   10 CONTINUE
C     WRITE(6,1013)LMARK
      MARK(LMARK)=0
      GO TO 104
C     *****
C     STEP 4
C     *****
  104 CONTINUE
      DO 12 I=1,NV
      IF(B(I,1).NE.0)GO TO 75
   12 CONTINUE
  115 CONTINUE
      IF(IOBJ.EQ.1)THEN
      IF(ZBAR.GT.ABC)THEN
      FEAS=0
      WRITE(6,1115)
      RETURN
      END IF
      WRITE(6,1341)
      WRITE(6,1440)ZBAR
 1440 FORMAT(//,10X,,***OPTIMAL SOLUTION WITH OBJECTIVE ,,
     & ,FUNCTION VALUE,,I8,/,10X,,IS OBTAINED , PATHS,,
     & , COVERED ARE GIVEN BELOW,)
      DO 1447 I=1,LBAR
      ISET=BLOCK(BBAR(I,2),BBAR(I,1))
      WRITE(6,211)ISET,AIR(ISET),C(ISET)
 1447 CONTINUE
C     CALL TIMER(BB)
      RETURN
      END IF
      IF(ZBAR.GT.ABC)THEN
      IF(MAXMIN.LT.ABC)THEN
      WRITE(6,1341)
 1341 FORMAT(//,10X,,***FINAL RESULTS OF THE RUN***,,//)
      WRITE(6,1342)MAXMIN
 1342 FORMAT(/,10X,,MINIMAX COST IS,,I8,/,10X,
     & ,THE SOLUTION THAT REALIZES THIS COST IS,)
      DO 125 I=1,KLBAR
      WRITE(6,211)IBOPT(I),AIR(IBOPT(I)),C(IBOPT(I))
  211 FORMAT(12X,,PATH NO,,I5,3X,,PORT,,
     & I2,2X,,TIME,,I8)
  125 CONTINUE
C     CALL TIMER(BB)
      RETURN
      END IF
      FEAS=0
      WRITE(6,1115)
 1115 FORMAT(//,10X,,***PROBLEM IS FINAL INFEASIBLE***,)
      RETURN
      END IF
C     CMAX=0
      IF(IPRINT.GE.1)THEN
      WRITE(6,100)ZBAR
  100 FORMAT(//,10X,,***CURRENT OPTIMAL SOLUTION WITH ,,
     & ,OBJECTIVE FUNCTION VALUE,,I8,/,10X,
     & ,IS OBTAINED AND PATHS COVERED ARE GIVEN BELOW,)
      DO 17 I=1,LBAR
      ISET=BLOCK(BBAR(I,2),BBAR(I,1))
C     IF(C(ISET).GT.CMAX)THEN
```

```
      KISET=ISET
      CMAX=C(ISET)
      END IF
      WRITE(6,211)ISET,AIR(ISET),C(ISET)
  200 FORMAT(20X,'SET NO',I5,5X,'PATH NO',I5)
   17 CONTINUE
      IF(CMAX.LT.MAXMIN)THEN
      MAXMIN=CMAX
      END IF
      WRITE(6,1042)KISET,CMAX
 1042 FORMAT(///,10X,'IN THIS SOLUTION PATH',I5,/,10X,'HAS THE ',
     & 'MAXIMUM COST OF',I8)
      END IF
C     ************************
C     BOTTLENECK FORMULATION
C     ************************
      OMAX=MAXMIN
      DO 57 I=1,NR
      DO 58 J=1,MAXA
      ISET=BLOCK(I,J)
      IF(ISET.EQ.0)GO TO 57
      IF(C(ISET).GE.OMAX)THEN
      BLOCK(I,J)=0
      NEB(I)=NEB(I)-1
      NP=NP-1
      END IF
   58 CONTINUE
   57 CONTINUE
      MAXA=0
      DO 59 I=1,NR
      IF(NEB(I).GT.MAXA)MAXA=NEB(I)
   59 CONTINUE
      IF(IPRINT.EQ.2)THEN
      WRITE(6,1045)OMAX
 1045 FORMAT(///,10X,'BOTTLENECK REDUCTION OF SETS WITH COSTS',/
     & ,10X,'GREATER THEN OR EQUQL TO',I8,/,10X,
     & 'RESULTED THE FOLLOWING,')
      N1=NR
      N2=MAXA
      CALL IMAT(BLOCK,NOTJZ,NBIN,LAE2)
      END IF
      IF(IPRINT.GE.1)THEN
      WRITE(6,234)MAXA
      WRITE(6,1046)
 1046 FORMAT(/,10X,'START A NEW ITERATION')
      END IF
      GO TO 101
   75 CONTINUE
      L=B(LBOY,2)
      K=B(LBOY,1)
      B(LBOY,1),B(LBOY,2)=0
      LBOY=LBOY-1
      ISET=BLOCK(L,K)
      Z=Z-C(ISET)
      AIRMED(AIR(ISET))=AIRMED(AIR(ISET))-1
C     WRITE(6,1007)ISET
 1007 FORMAT(/,10X,'AT STEP 4 : REMOVE REAL SET',I4)
C     WRITE(6,1010)K,L
 1010 FORMAT(15X,'K = ',I4,2X,'L = ',I4)
      DO 13 I=1,NR
      IF(ADJ(I,ISET).EQ.1)E(I)=0
   13 CONTINUE
C     WRITE(6,1011)(E(I),I=1,NR),Z
      P=L
      MARK(ISET)=0
C     WRITE(6,1013)ISET
 1013 FORMAT(10X,'DEMARK REAL SET',I4)
      ISET=BLOCK(L,K+1)
      IF(ISET.GT.0)GO TO 95
      IF(L.EQ.1)GO TO 115
      GO TO 104
   95 CONTINUE
      ICON=4
      MARK(ISET)=1
      LMARK=ISET
C     WRITE(6,1012)ISET
      GO TO 103
C     ******
C     STEP 5
C     ******
  105 CONTINUE
      LBOY=LBOY+1
      B(LBOY,1)=J
      B(LBOY,2)=P
      ISET=BLOCK(P,J)
      WRITE(6,1006)ISET,J,P
 1006 FORMAT(/,10X,'AT STEP 5 : REAL SET',I4,2X,'IS TAKEN',
     & /,15X,'FROM REL SET',I4,2X,'OF BLOCK',I4)
      DO 14 I=1,NR
      IF(ADJ(I,ISET).EQ.1)E(I)=1
   14 CONTINUE
      AIRMED(AIR(ISET))=AIRMED(AIR(ISET))+1
      Z=Z+C(ISET)
      WRITE(6,1011)(E(I),I=1,NR),Z
 1011 FORMAT(15X,'E : ',6I4,2X,'Z=',I8)
      MARK(LMARK)=0
      WRITE(6,1013)LMARK
      DO 15 I=1,NR
      IF(E(I).EQ.0)GO TO 102
   15 CONTINUE
      IF(ICONS.EQ.2)THEN
      DO 212 I=1,MAIR
      IF(AIRMED(I).NE.AIRMAX(I))GO TO 104
  212 CONTINUE
      END IF
      DO 16 I=1,LBOY
      BBAR(I,1)=B(I,1)
      BBAR(I,2)=B(I,2)
   16 CONTINUE
      ZBAR=Z
      LBAR=LBOY
      CMAX=0
      ITFEAS=ITFEAS+1
      IF(IPRINT.GE.1)THEN
      WRITE(6,300)ITFEAS,Z
  300 FORMAT(/,10X,'FEASIBLE SOLUTION NO',I5,2X,'WITH',
     & ' OBJECTIVE FUNCTION VALUE',I8,/,10X,
     & 'GIVEN BELOW')
      END IF
      DO 18 I=1,LBOY
      ISET=BLOCK(B(I,2),B(I,1))
      IF(C(ISET).GT.CMAX)THEN
      CMAX=C(ISET)
      KISET=ISET
      END IF
      IF(IPRINT.GE.1)THEN
      WRITE(6,211)ISET,AIR(ISET),C(ISET)
      END IF
   18 CONTINUE
      IF(IPRINT.GE.1)THEN
      WRITE(6,1042)KISET,CMAX
      END IF
      IF(CMAX.LT.MAXMIN)THEN
      MAXMIN=CMAX
      DO 86 I=1,LBAR
      IBOPT(I)=BLOCK(B(I,2),B(I,1))
   86 CONTINUE
      KLBAR=LBAR
      END IF
      GO TO 104
 1385 CONTINUE
      WRITE(6,1310)
 1310 FORMAT(//,10X,'***BLOCK SIZE EXCEEDS PROGRAM LIMITS***')
      RETURN
      END
THESIS7DES-MTSP
```

```
PROGRAMS OF ,MINIMUM TOTAL COST ROUTING OF MILITARY CARGO AIRCRAFT,
-----------------------------------------------------------------------

PREPARED BY : MURAT KASAROGLU

DEFINITIONS OF PROGRAM LIMITS :
-------------------------------

KMAX   =MAXIMUM NUMBER OF PORTS
PLMAX  =MAXIMUM NUMBER OF PLANES
NMAX   =MAXIMUM SIZE OF W
LMAX   =MAXIMUM NUMBER OF LOADS

DEFINITIONS OF VARIABLES USED IN PROGRAMS AND INPUT DATA
--------------------------------------------------------

DMAX   =MAXIMUM LENGHT OF PATHS TO BE GENERATED
NPORT  =NUMBER OF PORTS
NLOAD  =NUMBER OF LOADS
NAIR   =NUMBER OF PLANES
NDUM   =TOTAL NUMBER OF DUMMIES
NPP    =NUMBER OF PORTS WITH PLANES INITIALLY
NN     =NPP+NLOAD (SIZE OF FINAL MATRIX W)
NNODE  =NN+2
       :TOTAL NUMBER OF NODES ON FORMULATED W.
D      =PORT TO PORT FLIGHT TIME MATRIX
       :D(I,J)=FLIGHT TIME BETWEEN PORT I AND PORT J
LOAD   =LOADS BETWEEN PORTS
       :LOAD(I,1)=STARTING PORT OF I,TH LOAD
       :LOAD(I,2)=ENDING   PORT OF I,TH LOAD
MAIR   =NUMBER OF PLANES AT EACH PORT
       :MAIR(I)=NUMBER OF PLANES AT PORT I
TLOAD  =TIME OF LOADING AND UNLOADING
W      =FORMULATED MATRIX ON WHICH PATHS ARE GENERATED


PROGRAMS RELATED WITH BOTTLENECK ROUTING :
------------------------------------------

THESIS-14: FORMULATE MDMTSP SO THAT TO MINIMIZE TOTAL FLIGHT TIME
THESIS-15: FORMULATE MDMTSP SO THAT TO MINIMIZE TOTAL FLIGHT TIME
           BY TRYING TO USE LESS PLANES
THESIS-16: FORMULATE MDMTSP SO THAT TO MINIMIZE TOTAL FLIGHT TIME
           AND MINIMIZE THE USE OF PLANES
THESIS-17: SOLVE TRAVELLING SALESMAN PROBLEM BY USING BRANCH AND BO


HESIS-14
```

```
      ************************************************************
      FORMULATE AND SOLVE A MULTI DEPOT MULTI SALESMAN
      TRAVELLING SALESMAN PROBLEM
      IN ORDER TO SOLVE MILITARY CARGO AIRCRAFT ROUTING PROBLEMS
      SO THAT TO MINIMIZE TOTAL MISSION TIME
      BY USING EXACTLY GIVEN NUMBER OF PLANES

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


      IMPLICIT INTEGER (A-Z)
      PARAMETER KMAX=50
      PARAMETER LMAX=100
      PARAMETER NMAX=100
      DIMENSION D(KMAX,KMAX),LOAD(LMAX,2),MAIR(KMAX),W(NMAX,NMAX)
      DIMENSION TOUR(NMAX),DUM(KMAX*2)
      CHARACTER AA*50/,***SIZE OF PROGRAM EXCEEDED***,/
      CHARACTER BB*50/,***FORMULATED MTSP MATRIX***,/
      CHARACTER DD*50/,***PORT TO PORT DISTANCE MATRIX***,/
      CHARACTER EE*50/,MTSP FORMULATION HAS STARTED,/
      CHARACTER FF*50/,JOB COMPLETED,/
      COMMON /IMAT1/NR,NC
      PRINT 23000
 3000 FORMAT(1H1)
      CALL TIMER(EE)
      A=99999
      READ(5,*)NPORT
      DO 1 I=1,NPORT
    1 READ(5,*)(D(I,J),J=1,NPORT)
      READ(5,*)(MAIR(I),I=1,NPORT)
      READ(5,*)TLOAD
      I=0
   15 I=I+1
      READ(5,*,END=99,ERR=99)LOAD(I,1),LOAD(I,2)
      GO TO 15
   99 NLOAD=I-1
      NAIR=0
      DO 2 I=1,NPORT
    2 NAIR=NAIR+MAIR(I)
      NDUM=NAIR*2
      NN=NLOAD+NDUM
      NR,NC=NPORT
      CALL TMAT(D,KMAX,KMAX,DD)
      WRITE(6,1000)NPORT
 1000 FORMAT(/,10X,,NUMBER OF AIRPORTS :,,I4)
      WRITE(6,2000)NLOAD
 2000 FORMAT(/,10X,,NUMBER OF LOADS :,,I4)
      WRITE(6,8000)TLOAD
 8000 FORMAT(/,10X,,LOADING PLUS UNLOADING TIME :,,I4)
      WRITE(6,3000)NN
 3000 FORMAT(/,10X,,NUMBER OF NODES GENERATED :,,I4)
      WRITE(6,21000)
 1000 FORMAT(//,10X,,NUMBER OF PLANES AT AIRPORTS,,/,10X,28('*'),/)
      DO 61 I=1,NPORT
      WRITE(6,22000)I,MAIR(I)
 2000 FORMAT(10X,,PORT,,I5,10X,,AVAILABLE PLANE,,I5)
   61 CONTINUE
      WRITE(6,4000)
 4000 FORMAT(//,10X,,MEANINGS OF DUMMY NODES,,/,10X,28(,*,),/)
      LK=0
      DO 41 I=1,NPORT
      IF(MAIR(I).EQ.0)GO TO 41
      DO 42 J=1,MAIR(I)
      LK=LK+1
      DUM(LK)=I
      WRITE(6,5000)LK,I,J
      LK=LK+1
      DUM(LK)=I
      WRITE(6,6000)LK,I,J
 5000 FORMAT(10X,,NODE,,I5,5X,,: PORT,,I5,5X,,PLANE,,I5,5X,,DEPARTURE,)
 6000 FORMAT(10X,,NODE,,I5,5X,,: PORT,,I5,5X,,PLANE,,I5,5X,,ARRIVAL  ,)
   42 CONTINUE
   41 CONTINUE
      DO 43 I=1,NLOAD
      LK=LK+1
      WRITE(6,7000)LK,LOAD(I,1),LOAD(I,2)
 7000 FORMAT(10X,,NODE,,I5,5X,,: LOAD FROM,I5,5X,,TO,,I5)
   43 CONTINUE
      IF(NN.GT.NMAX)THEN
      CALL TIMER(AA)
      STOP
      END IF
C
C     ***************************
C     FORMULATE THE MULTI DEPOT MTSP
C     ***************************
C
C     DUMMY TO DUMMY (PORT TO PORT)
C     **************************
      DO 3 I=1,NDUM
      DO 3 J=1,NDUM
    3 W(I,J)=A
      DO 4 L=2,NDUM-1,2
      K=L+1
      W(L,K)=0
    4 CONTINUE
      W(NDUM,1)=0
C     PORT TO LOAD
C     ***********
      DO 5 L=2,NDUM,2
      DO 5 K=NDUM+1,NN
    5 W(L,K)=A
C     LOAD TO PORT
C     ***********
      DO 6 K=1,NDUM-1,2
      DO 6 L=NDUM+1,NN
    6 W(L,K)=A
C     PORT TO LOAD
C     ***********
      L=-1
      DO 7 I=1,NPORT
      IF(MAIR(I).EQ.0)GO TO 7
      DO 8 J=1,MAIR(I)
      L=L+2
      DO 9 K=NDUM+1,NN
      KK=K-NDUM
      W(L,K)=D(LOAD(KK,1),LOAD(KK,2))+TLOAD
      IF(LOAD(KK,1).EQ.I)GO TO 9
      W(L,K)=W(L,K)+D(I,LOAD(KK,1))
    9 CONTINUE
    8 CONTINUE
    7 CONTINUE
C     LOAD TO PORT
C     ***********
      K=0
      DO 10 I=1,NPORT
      IF(MAIR(I).EQ.0)GO TO 10
      DO 11 J=1,MAIR(I)
      K=K+2
      DO 12 L=NDUM+1,NN
      LL=L-NDUM
      W(L,K)=0
C     IF(LOAD(LL,2).EQ.I)GO TO 12
C     W(L,K)=D(LOAD(LL,2),I)
   12 CONTINUE
   11 CONTINUE
   10 CONTINUE
C     LOAD TO LOAD
C     ***********
      DO 21 I=1,NLOAD
      DO 22 J=1,NLOAD
      L=I+NDUM
      K=J+NDUM
      IF(I.EQ.J)THEN
      W(L,K)=A
      GO TO 22
      END IF
      W(L,K)=D(LOAD(J,1),LOAD(J,2))+TLOAD
      IF(LOAD(I,2).EQ.LOAD(J,1))GO TO 22
      W(L,K)=W(L,K)+D(LOAD(I,2),LOAD(J,1))
   22 CONTINUE
   21 CONTINUE
      NR,NC=NN
      CALL TMAT(W,NMAX,NMAX,BB)
      IPRINT=1
C     SOLVE TSP
C     *********
      CALL TSP3(W,NN,A,TOUR,COST,IPRINT)
C     INTERPRET THE TSP TOUR
C  \  *********************
      PRINT 14000
```

```
4000 FORMAT(///,10X,,OPTIMAL ROUTES OF PLANES,,/,10X,24(,*,),/)
      LL=0
      DO 51 I=1,NN-1
      IF(TOUR(I).LE.NDUM)THEN
      IF(I.GT.1)THEN
      IF(TOUR(I-1).GT.NDUM)THEN
      OPORT=DUM(TOUR(I))
      IF(OPORT.NE.PORT)WRITE(6,27000)OPORT
7000  FORMAT(10X,,DESTINATION PORT FOR THIS PLANE IS :,,I4,
     & /,10X,,WHICH IS INCOMPATIBLE WITH ORIGIN,)
      END IF
      END IF
      IF(TOUR(I+1).GT.NDUM)THEN
      PORT=DUM(TOUR(I))
      LL=LL+1
      WRITE(6,25000)
5000  FORMAT(10X,50(,-,))
      WRITE(6,11000)LL,PORT
1000  FORMAT(/,10X,,PLANE NO :,,I4,5X,,ORIGINATING FROM PORT :,,I4,
     & /,10X,,IS ASSIGNED TO FOLLOWING ROUTE,)
      END IF
      GO TO 51
      END IF
      NL=TOUR(I)-NDUM
      WRITE(6,12000)TOUR(I),LOAD(NL,1),LOAD(NL,2)
2000  FORMAT(10X,,NODE,,I5,2X,,:    FROM PORT,,I5,5X,,TO PORT,,I5)
  51  CONTINUE
      OPORT=DUM(TOUR(NN))
      IF(OPORT.NE.PORT)WRITE(6,27000)OPORT
      WRITE(6,25000)
      WRITE(6,28000)LL,COST
8000  FORMAT(///,10X,,OPTIMAL TOTAL COST WITH,,I4,3X,,PLANES IS,,I11,//)
      CALL TIMER(FF)
      STOP
      END
HESIS-15
```

```
C
C
C     ************************************************************
C
C     FORMULATE AND SOLVE A MULTI DEPOT MULTI SALESMAN
C     TRAVELLING SALESMAN PROBLEM
C     IN ORDER TO SOLVE MILITARY CARGO AIRCRAFT ROUTING PROBLEMS
C     SO THAT TO MINIMIZE TOTAL MISSION TIME
C     BY USING GIVEN OR LESS NUMBER OF PLANES
C
C
C     ************************************************************
C
      IMPLICIT INTEGER (A-Z)
      PARAMETER KMAX=50
      PARAMETER LMAX=100
      PARAMETER NMAX=100
      DIMENSION D(KMAX,KMAX),LOAD(LMAX,2),MAIR(KMAX),W(NMAX,NMAX)
      DIMENSION TOUR(NMAX),DUM(KMAX*2)
      CHARACTER AA*50/,***SIZE OF PROGRAM EXCEEDED***,/
      CHARACTER BB*50/,***FORMULATED MTSP MATRIX***,/
      CHARACTER DD*50/,***PORT TO PORT DISTANCE MATRIX***,/
      CHARACTER EE*50/,MTSP FORMULATION HAS STARTED,/
      CHARACTER FF*50/,JOB COMPLETED,/
      COMMON /IMAT1/NR,NC
      PRINT 23000
23000 FORMAT(1H1)
      CALL TIMER(EE)
      NA=99999
      READ(5,*)NPORT
      DO 1 I=1,NPORT
    1 READ(5,*)(D(I,J),J=1,NPORT)
      READ(5,*)(MAIR(I),I=1,NPORT)
      READ(5,*)TLOAD
      I=0
   15 I=I+1
      READ(5,*,END=99,ERR=99)LOAD(I,1),LOAD(I,2)
      GO TO 15
   99 NLOAD=I-1
      NAIR=0
      DO 2 I=1,NPORT
    2 NAIR=NAIR+MAIR(I)
      NDUM=NAIR*2
      NN=NLOAD+NDUM
      NR,NC=NPORT
      CALL TMAT(D,KMAX,KMAX,DD)
      WRITE(6,1000)NPORT
 1000 FORMAT(/,10X,,NUMBER OF AIRPORTS :,,I4)
      WRITE(6,2000)NLOAD
 2000 FORMAT(/,10X,,NUMBER OF LOADS :,,I4)
      WRITE(6,8000)TLOAD
 8000 FORMAT(/,10X,,LOADING PLUS UNLOADING TIME :,,I4)
      WRITE(6,3000)NN
 3000 FORMAT(/,10X,,NUMBER OF NODES GENERATED :,,I4)
      WRITE(6,21000)
21000 FORMAT(//,10X,,NUMBER OF PLANES AT AIRPORTS,,/,10X,28(,*,),/)
      DO 61 I=1,NPORT
      WRITE(6,22000)I,MAIR(I)
22000 FORMAT(10X,,PORT,,I5,10X,,AVAILABLE PLANE,,I5)
   61 CONTINUE
      WRITE(6,4000)
 4000 FORMAT(//,10X,,MEANINGS OF DUMMY NODES,,/,10X,28(,*,),/)
      LK=0
      DO 41 I=1,NPORT
      IF(MAIR(I).EQ.0)GO TO 41
      DO 42 J=1,MAIR(I)
      LK=LK+1
      DUM(LK)=I
      WRITE(6,5000)LK,I,J
      LK=LK+1
      DUM(LK)=I
      WRITE(6,6000)LK,I,J
 5000 FORMAT(10X,,NODE,,I5,5X,,: PORT,,I5,5X,,PLANE,,I5,5X,,DEPARTURE,)
 6000 FORMAT(10X,,NODE,,I5,5X,,: PORT,,I5,5X,,PLANE,,I5,5X,,ARRIVAL   ,)
   42 CONTINUE
   41 CONTINUE
      DO 43 I=1,NLOAD
      LK=LK+1
      WRITE(6,7000)LK,LOAD(I,1),LOAD(I,2)
 7000 FORMAT(10X,,NODE,,I5,5X,,: LOAD FROM,I5,5X,,TO,,I5)
```

```
43  CONTINUE
    IF(NN.GT.NMAX)THEN
    CALL TIMER(AA)
    STOP
    END IF

    ******************************
    FORMULATE THE MULTI DEPOT MTSP
    ******************************

    DUMMY TO DUMMY (PORT TO PORT)
    *****************************
    DO 3 I=1,NDUM
    DO 3 J=1,NDUM
    W(I,J)=A
    IF(I+1.EQ.J)W(I,J)=0
 3  CONTINUE
    DO 4 L=2,NDUM-1,2
    K=L+1
    W(L,K)=0
 4  CONTINUE
    W(NDUM,1)=0
    PORT TO LOAD
    ************
    DO 5 L=2,NDUM,2
    DO 5 K=NDUM+1,NN
 5  W(L,K)=A
    LOAD TO PORT
    ************
    DO 6 K=1,NDUM-1,2
    DO 6 L=NDUM+1,NN
 6  W(L,K)=A
    PORT TO LOAD
    ************
    L=-1
    DO 7 I=1,NPORT
    IF(MAIR(I).EQ.0)GO TO 7
    DO 8 J=1,MAIR(I)
    L=L+2
    DO 9 K=NDUM+1,NN
    KK=K-NDUM
    W(L,K)=D(LOAD(KK,1),LOAD(KK,2))+TLOAD
    IF(LOAD(KK,1).EQ.I)GO TO 9
    W(L,K)=W(L,K)+D(I,LOAD(KK,1))
 9  CONTINUE
 8  CONTINUE
 7  CONTINUE
    LOAD TO PORT
    ************
    K=0
    DO 10 I=1,NPORT
    IF(MAIR(I).EQ.0)GO TO 10
    DO 11 J=1,MAIR(I)
    K=K+2
    DO 12 L=NDUM+1,NN
    LL=L-NDUM
    W(L,K)=0
    IF(LOAD(LL,2).EQ.I)GO TO 12
    W(L,K)=D(LOAD(LL,2),I)
12  CONTINUE
11  CONTINUE
10  CONTINUE
    LOAD TO LOAD
    ************
    DO 21 I=1,NLOAD
    DO 22 J=1,NLOAD
    L=I+NDUM
    K=J+NDUM
    IF(I.EQ.J)THEN
    W(L,K)=A
    GO TO 22
    END IF
    W(L,K)=D(LOAD(J,1),LOAD(J,2))+TLOAD
    IF(LOAD(I,2).EQ.LOAD(J,1))GO TO 22
    W(L,K)=W(L,K)+D(LOAD(I,2),LOAD(J,1))
22  CONTINUE
21  CONTINUE
    NR,NC=NN
    CALL TMAT(W,NMAX,NMAX,BB)
    IPRINT=1
    SOLVE TSP
    *********
```

```
      CALL TSP3(W,NN,A,TOUR,COST,IPRINT)
C     INTERPRET THE TSP TOUR
C     **********************
      PRINT 14000
14000 FORMAT(///,10X,'OPTIMAL ROUTES OF PLANES',/,10X,24('*'),/)
      LL=0
      DO 51 I=1,NN-1
      IF(TOUR(I).LE.NDUM)THEN
      IF(I.GT.1)THEN
      IF(TOUR(I-1).GT.NDUM)THEN
      OPORT=DUM(TOUR(I))
      IF(OPORT.NE.PORT)WRITE(6,27000)OPORT
27000 FORMAT(10X,'DESTINATION PORT FOR THIS PLANE IS :',I4,
     & /,10X,'WHICH IS INCOMPATIBLE WITH ORIGIN')
      END IF
      END IF
      IF(TOUR(I+1).GT.NDUM)THEN
      PORT=DUM(TOUR(I))
      LL=LL+1
      WRITE(6,25000)
25000 FORMAT(10X,50('-'))
      WRITE(6,11000)LL,PORT
11000 FORMAT(/,10X,'PLANE NO :',I4,5X,'ORIGINATING FROM PORT :',I4,
     & /,10X,'IS ASSIGNED TO FOLLOWING ROUTE')
      END IF
      GO TO 51
      END IF
      NL=TOUR(I)-NDUM
      WRITE(6,12000)TOUR(I),LOAD(NL,1),LOAD(NL,2)
12000 FORMAT(10X,'NODE',I5,2X,':    FROM PORT',I5,5X,'TO PORT',I5)
   51 CONTINUE
      OPORT=DUM(TOUR(NN))
      IF(OPORT.NE.PORT)WRITE(6,27000)OPORT
      WRITE(6,25000)
      WRITE(6,28000)LL,COST
28000 FORMAT(///,10X,'OPTIMAL TOTAL COST WITH',I4,3X,'PLANES IS',I11,//)
      CALL TIMER(FF)
      STOP
      END
.THESIS-16
```

```
C     ****************************************************************
C     FORMULATE AND SOLVE A MULTI DEPOT MULTI SALESMAN
C     TRAVELLING SALESMAN PROBLEM
C     IN ORDER SOLVE MILITARY CARGO AIRCRAFT ROUTING PROBLEMS
C     SO THAT TO MINIMIZE TOTAL MISSION TIME
C     AND TO MINIMIZE THE COST OF USING EXCESS PLANES
C
C     ****************************************************************
      IMPLICIT INTEGER (A-Z)
      PARAMETER KMAX=50
      PARAMETER LMAX=100
      PARAMETER NMAX=100
      DIMENSION PCOST(KMAX)
      DIMENSION D(KMAX,KMAX),LOAD(LMAX,2),MAIR(KMAX),W(NMAX,NMAX)
      DIMENSION TOUR(NMAX),DUM(KMAX*2)
      CHARACTER AA*50/'***SIZE OF PROGRAM EXCEEDED***'/
      CHARACTER BB*50/'***FORMULATED MTSP MATRIX***'/
      CHARACTER DD*50/'***PORT TO PORT DISTANCE MATRIX***'/
      CHARACTER EE*50/'MTSP FORMULATION HAS STARTED'/
      CHARACTER FF*50/'JOB COMPLETED'/
      COMMON /IMAT1/NR,NC
      PRINT 23000
23000 FORMAT(1H1)
      CALL TIMER(EE)
      A=999999
      READ(5,*)NPORT
      DO 1 I=1,NPORT
    1 READ(5,*)(D(I,J),J=1,NPORT)
      READ(5,*)(MAIR(I),I=1,NPORT)
      READ(5,*)TLOAD
      I=0
   15 I=I+1
      READ(5,*,END=99,ERR=99)LOAD(I,1),LOAD(I,2)
      GO TO 15
   99 NLOAD=I-1
      READ(5,*)(PCOST(I),I=1,NPORT)
      NAIR=0
      DO 2 I=1,NPORT
    2 NAIR=NAIR+MAIR(I)
      NDUM=NAIR*2
      NN=NLOAD+NDUM
      NR,NC=NPORT
      CALL TMAT(D,KMAX,KMAX,DD)
      WRITE(6,1000)NPORT
 1000 FORMAT(/,10X,'NUMBER OF AIRPORTS :',I4)
      WRITE(6,2000)NLOAD
 2000 FORMAT(/,10X,'NUMBER OF LOADS :',I4)
      WRITE(6,8000)TLOAD
 8000 FORMAT(/,10X,'LOADING PLUS UNLOADING TIME :',I4)
      WRITE(6,3000)NN
 3000 FORMAT(/,10X,'NUMBER OF NODES GENERATED :',I4)
      WRITE(6,21000)
21000 FORMAT(//,10X,'NUMBER OF PLANES AT AIRPORTS',/,10X,28('*'),/)
      DO 61 I=1,NPORT
      WRITE(6,22000)I,MAIR(I),PCOST(I)
22000 FORMAT(10X,'PORT',I5,10X,'AVAILABLE PLANE',I5,
     & '  FIXED COST OF A PLANE AT THIS PORT',I6)
   61 CONTINUE
      WRITE(6,4000)
 4000 FORMAT(//,10X,'MEANINGS OF DUMMY NODES',/,10X,28('*'),/)
      LK=0
      DO 41 I=1,NPORT
      IF(MAIR(I).EQ.0)GO TO 41
      DO 42 J=1,MAIR(I)
      LK=LK+1
      DUM(LK)=I
      WRITE(6,5000)LK,I,J
      LK=LK+1
      DUM(LK)=I
      WRITE(6,6000)LK,I,J
 5000 FORMAT(10X,'NODE',I5,5X,': PORT',I5,5X,'PLANE',I5,5X,'DEPARTURE')
 6000 FORMAT(10X,'NODE',I5,5X,': PORT',I5,5X,'PLANE',I5,5X,'ARRIVAL  ')
   42 CONTINUE
   41 CONTINUE
      DO 43 I=1,NLOAD
      LK=LK+1
      WRITE(6,7000)LK,LOAD(I,1),LOAD(I,2)
 7000 FORMAT(10X,'NODE',I5,5X,': LOAD FROM',I5,5X,'TO',I5)
   43 CONTINUE
      IF(NN.GT.NMAX)THEN
      CALL TIMER(AA)
      STOP
      END IF
C
C     *****************************
C     FORMULATE THE MULTI DEPOT MTSP
C     *****************************
C
C     DUMMY TO DUMMY (PORT TO PORT)
C     *****************************
      DO 3 I=1,NDUM
      DO 3 J=1,NDUM
      W(I,J)=A
      IF(I+1.EQ.J)W(I,J)=0
    3 CONTINUE
      DO 4 L=2,NDUM-1,2
      K=L+1
      W(L,K)=0
    4 CONTINUE
      W(NDUM,1)=0
C     PORT TO LOAD
C     ***********
      DO 5 L=2,NDUM,2
      DO 5 K=NDUM+1,NN
    5 W(L,K)=A
C     LOAD TO PORT
C     ***********
      DO 6 K=1,NDUM-1,2
      DO 6 L=NDUM+1,NN
    6 W(L,K)=A
C     PORT TO LOAD
C     ***********
      L=-1
      DO 7 I=1,NPORT
      IF(MAIR(I).EQ.0)GO TO 7
      DO 8 J=1,MAIR(I)
      L=L+2
      DO 9 K=NDUM+1,NN
      KK=K-NDUM
      W(L,K)=D(LOAD(KK,1),LOAD(KK,2))+TLOAD+PCOST(I)
      IF(LOAD(KK,1).EQ.I)GO TO 9
      W(L,K)=W(L,K)+D(I,LOAD(KK,1))
    9 CONTINUE
    8 CONTINUE
    7 CONTINUE
C     LOAD TO PORT
C     ***********
      K=0
      DO 10 I=1,NPORT
      IF(MAIR(I).EQ.0)GO TO 10
      DO 11 J=1,MAIR(I)
      K=K+2
      DO 12 L=NDUM+1,NN
      LL=L-NDUM
      W(L,K)=0
C     IF(LOAD(LL,2).EQ.I)GO TO 12
C     W(L,K)=D(LOAD(LL,2),I)
   12 CONTINUE
   11 CONTINUE
   10 CONTINUE
C     LOAD TO LOAD
C     ***********
      DO 21 I=1,NLOAD
      DO 22 J=1,NLOAD
      L=I+NDUM
      K=J+NDUM
      IF(I.EQ.J)THEN
      W(L,K)=A
      GO TO 22
      END IF
      W(L,K)=D(LOAD(J,1),LOAD(J,2))+TLOAD
      IF(LOAD(I,2).EQ.LOAD(J,1))GO TO 22
      W(L,K)=W(L,K)+D(LOAD(I,2),LOAD(J,1))
   22 CONTINUE
   21 CONTINUE
      NR,NC=NN
      CALL TMAT(W,NMAX,NMAX,BB)
      IPRINT=1
C     SOLVE TSP
C     *********
      CALL TSP3(W,NN,A,TOUR,COST,IPRINT)
C     INTERPRET THE TSP TOUR
```

```
C     **********************
      PRINT 14000
14000 FORMAT(///,10X,,OPTIMAL ROUTES OF PLANES,,/,10X,24(,*,),/)
      LL=0
      DO 51  I=1,NN-1
      IF(TOUR(I).LE.NDUM)THEN
      IF(I.GT.1)THEN
      IF(TOUR(I-1).GT.NDUM)THEN
      OPORT=DUM(TOUR(I))
      IF(OPORT.NE.PORT)WRITE(6,27000)OPORT
27000 FORMAT(10X,,DESTINATION PORT FOR THIS PLANE IS :,,I4,
    & /,10X,,WHICH IS INCOMPATIBLE WITH ORIGIN,)
      END IF
      END IF
      IF(TOUR(I+1).GT.NDUM)THEN
      PORT=DUM(TOUR(I))
      LL=LL+1
      WRITE(6,25000)
25000 FORMAT(10X,50(,-,))
      WRITE(6,11000)LL,PORT
11000 FORMAT(/,10X,,PLANE NO :,,I4,5X,,ORIGINATING FROM PORT :,,I4,
    & /,10X,,IS ASSIGNED TO FOLLOWING ROUTE,)
      END IF
      GO TO 51
      END IF
      NL=TOUR(I)-NDUM
      WRITE(6,12000)TOUR(I),LOAD(NL,1),LOAD(NL,2)
12000 FORMAT(10X,,NODE,,I5,2X,,:   FROM PORT,,I5,5X,,TO PORT,,I5)
   51 CONTINUE
      OPORT=DUM(TOUR(NN))
      IF(OPORT.NE.PORT)WRITE(6,27000)OPORT
      WRITE(6,25000)
      WRITE(6,28000)LL,COST
28000 FORMAT(///,10X,,OPTIMAL TOTAL COST WITH,,I4,3X,,PLANES IS,,I11,//)
      CALL TIMER(FF)
      STOP
      END
).THESIS-17
```

```
      SUBROUTINE TSP3(ICM,IBOY,IAAAA,ITOUR,ITCOST,IPRINT)
C*********************************************************************
C*********************************************************************
C*********************************************************************
C
C     THIS SUBROUTINE APPLIES BRANCH AND BOUND
C     ALGORITHM TO SOLVE TRAVELING SALESMAN PROBLEMS
C
C
C*********************************************************************
C*********************************************************************
C*********************************************************************
      PARAMETER NNNN=100
      DIMENSION IRSIL(NNNN),ICSIL(NNNN),ICM(NNNN,NNNN),IDOL(NNNN),
     *KTOUR(NNNN),IYOL(NNNN),JYOL(NNNN),LBFREE(NNNN),LBFIX(NNNN),
     *   IC(NNNN),IA(NNNN),IB(NNNN),ICN(NNNN,NNNN)
      DIMENSION ITOUR(NNNN)
      CHARACTER*50 A/,A TSP TOUR HAS OBTAINED,/
      CHARACTER*50 AA/,TSP HAS STARTED,/
      CHARACTER*50 AAA/,TSP HAS FINNED,/
C     DEFINE ICN(I,K)=BITS(IC(I,K),1,18)
C     DEFINE ICM(I,L)=BITS(IC(I,L),19,18)
  567 FORMAT(1H1)
      CALL TIMER(AA)
      ITCON=0
      DO 2 I=1,IBOY
      DO 2 J=1,IBOY
      ICN(I,J)=ICM(I,J)
    2 CONTINUE
      ICATAL=0
      ITUT=0
      IZ=IAAAA
      IUB=1
      IMS=0
  500 ICOUNT=0
C
C*********************************
C     ROW AND COLUMN REDUCTIONS
C*********************************
C
      DO 4 I=1,IBOY
      MIN=IAAAA
      IF(IRSIL(I).EQ.1)GO TO 4
      DO 5 J=1,IBOY
      IF(ICSIL(J).EQ.1)GO TO 5
      IF(ICM(I,J).GE.MIN)GO TO 5
      MIN=ICM(I,J)
    5 CONTINUE
      IDOL(I)=MIN
      DO 6 J=1,IBOY
      IF(ICSIL(J).EQ.1)GO TO 6
      ICM(I,J)=ICM(I,J)-MIN
    6 CONTINUE
    4 CONTINUE
      ISUM1=0
      DO 10 I=1,IBOY
      ISUM1=ISUM1+IDOL(I)
   10 CONTINUE
      DO 7 J=1,IBOY
      MIN=IAAAA
      IF(ICSIL(J).EQ.1)GO TO 7
      DO 8 I=1,IBOY
      IF(IRSIL(I).EQ.1)GO TO 8
      IF(ICM(I,J).GE.MIN)GO TO 8
      MIN=ICM(I,J)
    8 CONTINUE
      IDOL(J)=MIN
      DO 9 I=1,IBOY
      IF(IRSIL(I).EQ.1)GO TO 9
      ICM(I,J)=ICM(I,J)-MIN
    9 CONTINUE
    7 CONTINUE
      ISUM2=0
      DO 11 J=1,IBOY
      ISUM2=ISUM2+IDOL(J)
   11 CONTINUE
      IHOH=ISUM1+ISUM2
  200 IF(IUB.NE.1)THEN
C
C*********************************
```

```
C
C      SETTING INFINITIES INTO MATRIX
C      IN ORDER TO PREVENT SUBLOOPS
C
*******************************
C
      DO 805 L=1,IUB-1
      IJAT=1
      IT=IYOL(L)
      IPP=IT
      KTOUR(IJAT)=IPP
      IJAT=IJAT+1
  807 DO 806 K=1,IUB-1
      IF(IYOL(K).EQ.IPP)THEN
      KTOUR(IJAT)=JYOL(K)
      IPP=JYOL(K)
      GO TO 809
      ENDIF
  806 CONTINUE
      ICM(IPP,IT)=IAAAA
      GO TO 805
  809 IJAT=IJAT+1
      GO TO 807
  805 CONTINUE
      ENDIF
      IDUMMY=0
C
*******************************
C      PENALTY CALCULATION
*******************************
C
      DO 151 I=1,IBOY
      IF(IRSIL(I).EQ.1)GO TO 151
      DO 15 J=1,IBOY
      IF(ICSIL(J).EQ.1)GO TO 15
       IF(ICM(I,J).EQ.0)THEN
      MIN1=IAAAA
      DO 16 JJ=1,IBOY
      IF(JJ.EQ.J)GO TO 16
      IF(ICSIL(JJ).EQ.1)GO TO 16
      IF(ICM(I,JJ).GT.MIN1)GO TO 16
      MIN1=ICM(I,JJ)
   16 CONTINUE
      MIN2=IAAAA
      DO 27 II=1,IBOY
      IF(IRSIL(II).EQ.1)GO TO 27
      IF(II.EQ.I)GO TO 27
      IF(ICM(II,J).GT.MIN2)GO TO 27
      MIN2=ICM(II,J)
   27 CONTINUE
      IF((MIN1+MIN2).GE.IDUMMY)THEN
      INROV=I
      JNCOL=J
      IDUMMY=MIN1+MIN2
      ENDIF
      ENDIF
  15   CONTINUE
 151   CONTINUE
C
*******************************
C      BOUND ON FREE NODE
*******************************
C
      IF(IUB.EQ.1)THEN
      LBFREE(IUB)=IHOH+IDUMMY
      ELSE
      LBFREE(IUB)=LBFIX(IUB-1)+IDUMMY
      ENDIF
      ICSIL(JNCOL)=1
      IRSIL(INROV)=1
      ICOUNT=ICOUNT+1
      ICM(JNCOL,INROV)=IAAAA
      IYOL(IUB)=INROV
      JYOL(IUB)=JNCOL
C
*******************************
C      ROW AND COLUMN REDUCTIONS
*******************************
C
      DO 41 I=1,IBOY
      MIN=IAAAA
      IF(IRSIL(I).EQ.1)GO TO 41
      DO 51 J=1,IBOY
      IF(ICSIL(J).EQ.1)GO TO 51
      IF(ICM(I,J).GE.MIN)GO TO 51
      MIN=ICM(I,J)
   51 CONTINUE
      IDOL(I)=MIN
      DO 61 J=1,IBOY
      IF(ICSIL(J).EQ.1)GO TO 61
      ICM(I,J)=ICM(I,J)-MIN
   61 CONTINUE
  41  CONTINUE
      ISUM1=0
      DO 84 I=1,IBOY
      IF(IRSIL(I).EQ.1)GO TO 84
      ISUM1=ISUM1+IDOL(I)
   84 CONTINUE
      DO 71 J=1,IBOY
      MIN=IAAA
      IF(ICSIL(J).EQ.1)GO TO 71
      DO 81 I=1,IBOY
      IF(IRSIL(I).EQ.1)GO TO 81
      IF(ICM(I,J).GE.MIN)GO TO 81
      MIN=ICM(I,J)
   81 CONTINUE
      IDOL(J)=MIN
      DO 83 I=1,IBOY
      IF(IRSIL(I).EQ.1)GO TO 83
      ICM(I,J)=ICM(I,J)-MIN
   83 CONTINUE
   71 CONTINUE
      ISUM2=0
      DO 85   J=1,IBOY
       IF(ICSIL(J).EQ.1)GO TO 85
      ISUM2=ISUM2+IDOL(J)
   85 CONTINUE
      IH=ISUM1+ISUM2
C
*****************************************
C     BOUND ON FIX NODE
*****************************************
C
      IF(IUB.EQ.1)THEN
      LBFIX(IUB)=IHOH+IH
      ELSE
      LBFIX(IUB)=LBFIX(IUB-1)+IH
      ENDIF
C
************************************
C
C     MATRIX IS TWO BY TWO
C
************************************
C
      IF((IBOY-ICOUNT).EQ.2)THEN
      DO 22 IK=1,IBOY
      IF(IRSIL(IK).EQ.1)GO TO 22
      DO 26 JK=1,IBOY
      IF(ICSIL(JK).EQ.1)GO TO 26
      IUB=IUB+1
      IYOL(IUB)=IK
      JYOL(IUB)=JK
      IF((IBOY-ICOUNT).NE.1)THEN
      DO 700 LX=1,IUB
      IJAT=1
      IT=IYOL(LX)
      IPP=IT
      KTOUR(IJAT)=IPP
      IJAT=IJAT+1
  701 DO 702 KZ=1,IUB
      IF(IYOL(KZ).EQ.IPP)THEN
        KTOUR(IJAT)=JYOL(KZ)
      IPP=JYOL(KZ)
      IF(IPP.EQ.IT)GO TO 703
      GO TO 704
      ENDIF
  702 CONTINUE
      GO TO 700
  704 IJAT=IJAT+1
      GO TO 701
  700 CONTINUE
      DO 252 LKJ=1,IUB-1
      IF(IK.EQ.IYOL(LKJ).OR.JK.EQ.JYOL(LKJ))THEN
      GO TO 703
      ENDIF
 252   CONTINUE
      GO TO 705
 703  IUB=IUB-1
      GO TO 25
 705  LBFREE(IUB)=IAAAA
      IH=0
      LBFIX(IUB)=LBFIX(IUB-1)+IH
      IYOL(IUB)=IK
      JYOL(IUB)=JK
      ICSIL(JK)=1
      IRSIL(IK)=1
      ICM(JK,IK)=IAAAA
      ICOUNT=ICOUNT+1
      GO TO 22
      ELSE
      LBFREE(IUB)=IAAAA
      IH=0
      LBFIX(IUB)=LBFIX(IUB-1)+IH
      IYOL(IUB)=IK
      JYOL(IUB)=JK
      ICSIL(JK)=1
      ICM(JK,IK)=IAAAA
      IRSIL(IK)=1
      ICOUNT=ICOUNT+1
      GO TO 254
      ENDIF
  25  CONTINUE
  22  CONTINUE
 254  IF(LBFIX(IUB).LT.IZ)THEN
C
C*****************************************
C
C     A TSP TOUR OBTAINED
C
C*****************************************
C
      IZ=LBFIX(IUB)
      IMS=IUB
      IJAT=1
      IPP=1
      ITOUR(IJAT)=1
      IJAT=IJAT+1
  82  DO 86 I=1,IMS
      IF(IYOL(I).EQ.IPP)THEN
      ITOUR(IJAT)=JYOL(I)
      IPP=JYOL(I)
      IF(IPP.EQ.1)THEN
      IF(IPPINT.EQ.1)THEN
      ITCOST=0
      DO 564 IE=1,IUB
      ITCOST=ITCOST+ICM(IYOL(IE),JYOL(IE))
 564     CONTINUE
      ITCON=ITCON+1
      WRITE(6,8472)ITCON,ITCOST
 8472 FORMAT(/,10X,'TSP TOUR NO',I5,10X,'WITH TOTAL COST',I10,10X,
     & /,10X,  'IS OBTAINED AND GIVEN BELOW')
      CALL PATOUT(ITOUR,IBOY+1,NNNN)
      ENDIF
      GOTO 92
      END IF
      GO TO 90
      ENDIF
  86  CONTINUE
  90  IJAT=IJAT+1
      GO TO 82
      ENDIF
      ENDIF
C
C*****************************************
C
C     CONTROL IF CURRENT FIX BOUND
C     GREATER THAN CURRENT SOLUTION
C
C*****************************************
C
  92  IF(LBFIX(IUB).GE.IZ)THEN
      IF(IUB.GE.(IBOY-2))THEN
      IUB=IMS
      ELSE
      IMS=IUB
      ENDIF
      DO 89 III=IJB,1,-1
      IF(LBFREE(III).LT.IZ)THEN
      IUB=III
      IMS=III-1
      IF(IUB.LT.ITUT)THEN
          ITAT=ICATAL
      DO 453 KJ=ITAT,1,-1
      IF(IC(KJ).LE.IUB)THEN
      ICATAL=KJ
        GO TO 871
       ELSE
        ICATAL=KJ
      IF(KJ.EQ.1)THEN
      ICATAL=0
      ENDIF
      ENDIF
  453 CONTINUE
 871      CONTINUE
        ITUT=IUB
      ELSE
      ITUT=IUB
      ENDIF
      GO TO 100
      ENDIF
   89 CONTINUE
      GO TO 99
C
*****************************
C
C     SETTING ORIGINAL MATRIX
C
*****************************
C
  100 DO 110 I=1,IBOY
      DO 110 J=1,IBOY
      ICM(I,J)=ICN(I,J)
  110 CONTINUE
      DO 101 I=1,IBOY
      IRSIL(I)=0
      ICSIL(I)=0
 101  CONTINUE
      ICATAL=ICATAL+1
      IA(ICATAL)=IYOL(IJB)
      IB(ICATAL)=JYOL(IJB)
      IC(ICATAL)=IUB
      DO 124 IPO=ICATAL,1,-1
      ITZ=IA(IPO)
      ITX=IB(IPO)
      ICM(ITZ,ITX)=IAAAA
  124 CONTINUE
      IF(IUB.EQ.1)GO TO 500
      IG=0
      ICOUNT=0
      DO 102 II=1,IUB-1
      IRSIL(IYOL(II))=1
      ICSIL(JYOL(II))=1
      ICOUNT=ICOUNT+1
      IG=IG+ICM(IYOL(II),JYOL(II))
      ICM(JYOL(II),IYOL(II))=IAAAA
  102 CONTINUE
C
*********************************
C     ROW AND COLUMN REDUCTIONS
*********************************
      DO 140 I=1,IBOY
      MIN=IAAAA
      IF(IRSIL(I).EQ.1)GO TO 140
      DO 141 J=1,IBOY
      IF(ICSIL(J).EQ.1)GO TO 141
      IF(ICM(I,J).GE.MIN)GO TO 141
      MIN=ICM(I,J)
  141 CONTINUE
      IDOL(I)=MIN
      DO 142 J=1,IBOY
      IF(ICSIL(J).EQ.1)GO TO 142
      ICM(I,J)=ICM(I,J)-MIN
  142 CONTINUE
  140 CONTINUE
      ISUM1=0
      DO 167 I=1,IBOY
       IF(IRSIL(I).EQ.1)GO TO 167
      ISUM1=ISUM1+IDOL(I)
 167  CONTINUE
      DO 143 J=1,IBOY
      MIN=IAAAA
      IF(ICSIL(J).EQ.1)GO TO 143
      DO 144 I=1,IBOY
      IF(IRSIL(I).EQ.1)GO TO 144
      IF(ICM(I,J).GE.MIN)GO TO 144
      MIN=ICM(I,J)
 144  CONTINUE
      IDOL(J)=MIN
      DO 145 I=1,IBOY
      IF(IRSIL(I).EQ.1)GO TO 145
      ICM(I,J)=ICM(I,J)-MIN
 145  CONTINUE
 143  CONTINUE
      ISUM2=0
      DO 168 J=1,IBOY
      IF(ICSIL(J).EQ.1)GO TO 168
      ISUM2=ISUM2+IDOL(J)
 168  CONTINUE
      IH=ISUM1+ISUM2
       BFIX(IDB-1)=IH+IG
      ELSE
      IDB=IDB+1
      ENDIF
      GO TO 200
  99  IF(IPRINT.EQ.1)THEN
      PRINT 1943
1943  FORMAT(//,10X,'NO FURTHER IMPROVEMENT IS POSSIBLE LAST TOUR,'
    & ' IS OPTIMAL,')
      ENDIF
      CALL TIMER(AAA)
      RETURN
      END
THESIS/LIST
```