# FOUR HEURISTIC SOLUTION PROCEDURES TO THE TRAVELLING SALESMAN PROBLEM AND AN APPLICATION TO THE MULTI-DEPOT VEHICLE ROUTING PROBLEM

by

Yasef Tovya
B.S. in I.E., Boğaziçi University, 1981



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

ACKNOWLEDGEMENT

I am greatly indebted to Doç. Dr. Gündüz ULUSOY, my instructor and thesis advisor for his invaluable guidance and practical discussions of the techniques and procedures used. Not only was he tireless in co-operating throughout all the aspects of the study but he also provided tangible long-term encouragement.

Yasef TOVYA

# ABSTRACT

This study consists of two parts. In the first part, four heuristic algorithms for solving the Travelling Salesman Problem (TSP) is developed. Given a graph, the first algorithm forms a subgraph in which the necessary conditions for the existence of a travelling salesman tour are satisfied. In case the subgraph does not contain any travelling salesman tour, Little's branch and bound algorithm is partially applied to the resultant cost matrix. The second algorithm, starts with the minimum cost assignment and ranks the assignment solutions in ascending costs by introducing subtour breaking constraints. The third algorithm produces some best achievable n-paths which start from a root node and end at some node incident to the root node. These paths are then completed to travelling salesman tours and the least cost tour is taken as the best achievable solution. A geometric approach to solving the TSP is described in the last algorithm. Starting with a partial tour, the algorithm determines which of the remaining nodes are to be inserted between which consecutive pair of nodes on the subtour and in what order. After all, a summary of computational results regarding both the efficiency and the computational effort of all the algorithms is presented.

In the second part, it is shown that the TSP can be applied to the Multi-Depot Vehicle Routing Problem (MDVRP) in which p vehicles located at m depots deliver products to demand nodes. The routing decision involves determining what route each vehicle will follow so that the total distance travelled is minimized subject to the condition that the demands are satisfied, and the vehicles return to their original depots. A transformation is applied to the MDVRP in order to formulate it as a TSP. The transformed graph includes two additional nodes for each vehicle where one serves as a departure node and the other serves as an arrival node for the depot at which that particular vehicle is initially located. By imposing the additional requirement that each demand node is visited by one and only one vehicle the solution to the original problem can be obtained by solving the TSP on the transformed graph. As a result, the algorithms developed in the first part of the study are applied to solve the MDVRP. Computational results reveal that, the computational effort needed for solving the TSP in a transformed graph is less than the computational effort needed for solving the TSP in a complete travelling salesman graph of the same size.

# KISA ÖZET

Bu çalışma iki bölümden oluşmaktadır. Birinci bölümde, Gezgin Satıcı Probleminin (GSP) çözümünde kullanılabilecek dört sezgisel algoritma geliştirilmiştir. Birinci algoritma, verilen serime ait maliyet matrisini indirgemekte ve içinde bir gezgin satıcı turunun varolabilmesi için gerek şartların sağlandığı bir alt serim oluşturmaktadır. Alt serimde herhangi bir gezgin satıcı turu bulunmadığı takdirde ise Little'in dal ve düğüm yöntemi indirgenmiş olan maliyet matrisine kısmî olarak uygulanmaktadır. İkinci algoritma çözüme önce maliyet matrisi üzerinde bir atama problemi çözerek başlar. Daha sonra, elde edilen alt turları parçalayan kısıtlar probleme eklenerek yeni bir atama problemi çözülür. Probleme kısıt ekleme ve çözme süreci bir gezgin satıcı turu oluşuncaya kadar devam eder. Üçüncü algoritma ise bir noktadan başlayarak bu başlangıç noktasına bağlanabilen noktalarda biten ve problemin içindeki her noktayı sadece bir kere ziyaret eden yollar oluşturur. Bu yollar gezgin satıcı turlarına tamamlanarak aralarından maliyeti en düşük olanı, bulunabilecek en iyi çözüm olarak seçilir. Dördüncü algoritmada GSP'nin çözümüne geometrik olarak yaklaşılmaktadır. Buna göre, kısmî bir tur ile başlayan algoritma, kısmî tura dahil olmayan noktaların bu tura nasıl ve hangi sıra ile ekleneceklerini bularak bir gezgin satıcı turu oluşturur. Algoritmalara ait verimlilik ve hesaplama karmaşıklığı ile

ilgili sonuçlar bilgisayar sonuçları ile birlikte ayrıca özetlenmektedir.

Çalışmanın ikinci kısmında ise GSP'nin Çok Depolu Taşıt Güzergahı Belirleme Problemine (ÇDTGBP) uygulanabileceği gösterilmiştir. Bu problemde p depoya dağıtılmış m adet taşıt burada depolanmış olan ürünleri istem noktalarına dağıtmaktadır. Buna göre her taşıt öyle bir güzergah izlemelidir ki toplam katedilen mesafe enazlanırken tüm istemler karşılanmış ve taşıtlar depolarına dönmüş olmalıdır. Verilen serime uygulanacak olan bir dönüşüm problemin GSP olarak çözülmesini sağlar. Serimde, depo noktaları elenirken her taşıt için bulundukları depolara karşı gelen kalkış ve varış noktaları yaratılır. Böylece serimdeki her noktanın yalnız bir defa ziyaret edileceği gözönüne alındığında ÇDTGBP'nin çözümü dönüştürülmüş serimde GSP'nin çözülmesi ile elde edilebilir. Sonuç olarak çalışmanın ilk bölümünde geliştirilen algoritmalar ÇDTGBP'nin çözülmesinde uygulanmıştır. Bilgisayarda elde edilen sonuçlar dönüştürülmüş olan bir serimi çözmek için gerekli olan çabanın aynı büyüklükteki tambağlı bir serimde GSP'ni çözmek için gerekli çabadan daha az olduğunu göstermiştir.

TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# I. INTRODUCTION

## 1.1 DESCRIPTION OF THE PROBLEM AND ITS COMPLEXITY

The Travelling Salesman Problem (TSP) has been a challenge that has attracted researchers who have wanted to derive an efficient solution method for the problem. The problem is formidable in the sense that the associated solution methods are quite difficult contrary to the simplicity of its statement. Mainly, it is a classical example which represents the challenge of the combinatorial optimization problems that have found a considerable interest up to now.

Consider a case in which there is a set of n cities which are to be visited by a salesman. The salesman, starting from a city is required to visit each of $(n-1)$ other cities before returning to the start. The problem is to design a route which minimizes the total distance travelled assuming that the distances between all city pairs are known.

One possible way of approaching to this problem, which is certain to give the correct answer is to enumerate all the possible tours and pick the shortest one. However, the complete enumeration of all the possible tours becomes a computationally impossible task even for problems with relatively small number of cities. As a matter of fact, even the fastest algorithms designed for solving this problem exactly require an inordinate amount of time.

Similar to most of the other combinatorial optimization problems the TSP falls into a category which is well known as NP-complete problems. The letters NP stand for "Nondeterministic Polynomial". The status of this category is uncertain in the sense that only exponential algorithms are known for the NP-complete problems. Neither an efficient algorithm for solving the problems has been developed, nor has it been proven that such algorithms do not exist. However, NP-complete problems have a remarkable property. That is, each problem in this category is efficiently (i.e. in a polinomial time) reducible to another NP-complete problem. Consequently, if any one of them has an efficient algorithm, then every NP-complete problem can be solved efficiently [1,2].

Not all situations to which the TSP is confined involve cost minimization. Instead of cost other measures of effectiveness may be substituted according to their applications. The problem is known to have wide applications in frequently encountered problems arising in practical situations. Among those problems are scheduling, sequencing, and vehicle routing problems which can be interpreted as a TSP with side constraints [3].

## 1.2 INTERPRETATION OF THE TSP AS A VEHICLE ROUTING PROBLEM

The Vehicle Routing Problem (VRP) involves the visiting of a set of required stops in a network by vehicles. In other words, the stops or alternatively the destinations with known requirements must be served with a fleet of vehicles stationed at some depot(s) in such a way as to minimize some objective. It is also required that all vehicles must start and finish at the depot(s) where they are initially located.

Although the structure of the VRP reveals that it is related to the physical delivery of goods, the delivery operation may be replaced by a collection, collection and/or delivery or some other operation which may not even be of physical nature. In fact, the VRP appears frequently in practical situations not directly related to the physical delivery. For example, service delivery, house call-tours of a doctor, preventive maintenance tours are all VRPs in which there are no physical delivery operations.

The TSP can be interpreted as a VRP with one depot and with one vehicle whose capacity exceeds total demand. That is, a vehicle is required to visit all the destinations once and only once before returning to the depot where it is located. However, the structure of the problem is changed considerably when more vehicles, more depots, different vehicle capacities and additional route restrictions are involved. As a matter of fact, the VRP is also an NP-complete problem for which no polynomially bounded algorithm has been developed [4,5].

## 1.3 EXTENSION TO THE MULTIPLE TSP

An extension of the TSP which has proven to be more appropriate for serving as a core problem to the VRP is the Multiple Travelling Salesman Problem (MTSP). In this problem, m salesmen are required to design m subtours in such a way that each destination is visited exactly once by exactly one salesman while the total distance travelled by all the salesmen is being minimized. As a result, the MTSP can be interpreted as the problem of routing a fleet of m vehicles from a single depot to many destinations with the condition that each routed

vehicle will return to the depot and each destination will be visited once and only once. The vehicle capacities are assumed to exceed the demand in any subtour that may be designed.

Similar to the TSP, the MTSP is an NP-complete problem. However, it has been shown that the solution to the MTSP is no more difficult than the solution to the TSP [6,7,8]. In addition, equivalent TSP formulations of the MTSP have been derived. The equivalence is obtained by creating m copies of the depot one for each vehicle. Each of these copies are connected to each destination exactly as the original depot. That is, the distances associated with each such pair of nodes are the same. However, there is no connection between any pair of the copies of the depot. In other words infinities are inserted in the associated elements of the transformed matrix so that the optimal travelling salesman tour in the expanded graph will never contain an arc connecting any pair of the copies. Once the TSP is solved for the expanded matrix, the copies are coalesced back into a single depot and consequently the travelling salesman tour is decomposed into m subtours.

## 1.4 EXTENSION TO THE MULTI-DEPOT VEHICLE ROUTING PROBLEM

A further extension of the VRP is to allow vehicles to reside at more than one depot. The problem is known as the Multi-Depot Vehicle Routing Problem (MDVRP). A vehicle fleet of m vehicles distributed to p depots is required to satisfy the demand at each destination. The routing decision involves the determination of what route each vehicle will follow so that the total distance travelled is minimized subject

to the constraints (i) that the demands are satisfied, (ii) each destination is visited once and only once and (iii) the vehicles return to their oniginal depots.

Although the VRP has attracted considerable attention, the MDVRP has not been studied widely yet and therefore is a promising area for further research. In the relevant literature, it has been stated that exact methods for solving the single depot VRP can be extended to the multi-depot case [4]. The principal exact methods for solving the single depot VRP are branch and bound techniques [5]. But only methods based on heuristic programming have appeared to be computationally feasible for solving large practical problems. As it has been reported by Golden, et.al [4] problems with about four depots and hundred destinations can be handled on a computer in less than 10 seconds.

The property that any NP-complete problem can be reduced to another NP-complete problem in polynomial time is of particular importance in this case. Once the problem is reduced to another NP-complete problem for which an efficient heuristic algorithm is developed, the algorithm can be used to solve the original problem. The TSP is one such problem which has been studied widely. There are several heuristic algorithms for solving the TSP efficiently. Moreover, these algorithms require less computation effort in comparison with the algorithms developed for solving the MDVRP. Fortunately, it is possible to model the MDVRP as a TSP by transforming the original graph into one for which the TSP can be solved. The transformed graph includes two additional nodes for each one of the m vehicles where one

serves only as a departure node and the other serves as an arrival node for the depot at which that particular vehicle is initially located. Each departure node is connected to another arrival node with zero cost. In addition, the departure nodes are connected to each destination node and each destination node is connected to the arrival nodes with exactly the same costs as the corresponding original depots. The connections between all pairs of destination nodes remain the same. The cost matrix is updated accordingly. The solution to the original problem can be obtained by first solving the TSP on the transformed graph and then coalescing all the departure and arrival nodes back into p depots, so that the travelling salesman tour is decomposed into m subtours. Note that not all vehicles have to be used as a result of this transformation.

## 1.5 IMPORTANCE OF THE POLYNOMIALLY BOUNDED ALGORITHMS

As the size of the graphs being examined increases the time needed for solving the TSP and the NP-complete problems increases exponentially. Growth of this kind can be described by a mathematical function such as $a^n$ where n is a number related to the problem size. In fact, n is the number of cities for the TSP. Many other functions exist which can be regarded as having the same property of exponential growth. Among them are $n^n$ and $n!$. On the other hand, there exists some mathematical functions of another kind which are known as polynomials. What distinguishes polynomials from exponential functions is that n does not appear in an exponent. Linear functions, functions such as $n^2$, $n^3$ and the sum of such functions are all suitable for

describing the polynomially bounded computation times. For small values of n, a polynomial function may exceed an exponential one but there always exists a value n beyond which the exponential function is greater. For sufficiently large values of n any exponential function overtakes and exceeds the polynomial functions [9].

It has been accepted that algorithms whose execution time increases exponentially as a function of the size of the problem are not of practical value. Algorithms of this kind are known to be inefficient. For sufficiently large problems, a polynomially bounded algorithm executed on even the slowest computer will find the answer sooner than an exponential time algorithm on the fastest computer. This can be best seen in Table 1.1 [10].

TABLE 1.1 - The Maximum Size of Problems Solvable in one Hour With Respect to the Developments in Computer Technology

| Function | Existing Computers | Computers that are 100 times fast | Computers that are 1000 times fast |
|---|---|---|---|
| $n$ | $n_1$ | $100n_1$ | $1000n_1$ |
| $n^2$ | $n_2$ | $10n_2$ | $31.6n_2$ |
| $n^3$ | $n_3$ | $4.64n_3$ | $10n_3$ |
| $n^5$ | $n_4$ | $2.5\ n_4$ | $3.98n_4$ |
| $2^n$ | $n_5$ | $n_5 + 6.64$ | $n_5 + 9.97$ |
| $3^n$ | $n_6$ | $n_6 + 4.19$ | $n_6 + 6.29$ |

As a result, we can conclude that algorithms with exponential growth will not benefit from the technological developments made on the computers. That is, even if the efficiency of the computers improves by

a factor 1000 the time required for solving an exponentially bounded algorithm will decrease by a fixed amount which differs slightly from the efficiency of the existing algorithms.

## 1.6 OUTLINES OF THE ALGORITHMS DEVELOPED FOR SOLVING THE TSP

Since the NP-complete problems and therefore the TSP have no efficient algorithms, a possible way of attack is to seek approximate solutions that are good even if they are not precisely optimal instead of expending further effort in seeking optimum solutions. The thesis focuses first on the development of four algorithms for solving the TSP efficiently. The study is based on reducing the computational work while the resulting solutions remain close to the exact optimal solution. Experimental results reveal that this objective is achieved efficiently. Finally, the algorithms are applied to the MDVRP.

The first algorithm developed for solving the TSP uses a tour building approach. First, the cost matrix is reduced. The minimum element of each row is found and subtracted from every element in that row. In the resultant matrix, the minimum element of each column is subtracted from every element in that column. As a result, the arcs corresponding to the zero elements in the resultant matrix comprise a subgraph whose node set is the same as the original graph. Then, the cost matrix is further reduced in order to ensure that the necessary conditions for the existence of a travelling salesman tour hold in the subgraph. In other words, the reduction continues until there exists a path between each pair of nodes (i.e. strong connectedness). The reduction process is further invoked so that given a pair of nodes i

and j, there exists a path either from i to j or from j to i (i.e. unilateral connectedness) when one of the nodes is removed from the subgraph. Once the necessary conditions hold, the algorithm searches for a travelling salesman tour in the subgraph. In case the subgraph does not possess any travelling salesman tour, Little's branch and bound algorithm [11] is applied to the resultant matrix until a feasible tour can be obtained.

The relation between the Assignment Problem (AP) and the TSP forms the basis of the second algorithm developed. Considering the fact that the travelling salesman tours correspond to extreme points of the assignment polytope, the algorithm starts with the minimum cost assignment and finds new solutions ranked in ascending cost until a travelling salesman tour is obtained. At each iteration a cutting plane which forces the assignment solution to form a tour is introduced. The main difference between this algorithm and the related ones in literature is that no branch and bound procedure is involved. However, the efficiency is similar in the sense that it depends on the total number of extreme points between the optimum travelling salesman solution and the minimum cost assignment. In fact, the larger the problem size is, the larger is the number of extreme points in between.

A dynamic programming type approach is presented in the third algorithm. First, the elements of the cost matrix $C_{ij}$ are subtracted from a large number in order to achieve triangle inequality (i.e. $C_{ij} < C_{ik} + C_{kj}$ $\forall i,k,j$). At each stage, the algorithm adds a new arc to the sequence so that the total cost of the elementary path formed by the arc sequence is maximal. Finally, the elementary paths are

completed to travelling salesman tours and the one with the maximum cost is selected as the best achievable solution.

A geometric approach to the TSP is presented in the fourth algorithm. Similar to the related approaches in the literature the algorithm starts with the convex hull or alternatively a partial tour. Then, the travelling salesman tour is obtained by successive sequencing of each of the remaining nodes between consecutive pair of nodes on the partial tour. In order to determine the node to be inserted the heights of the triangles whose bases are determined by the arcs through consecutive pair of nodes on the partial tour and whose third vertices are the remaining nodes are calculated. The node corresponding to the largest of these heights is chosen and inserted into the sequence. The algorithm terminates when a travelling salesman tour is obtained.

All of the four algorithms proved to work well on several test problems. For small size problems, it was possible to check the difference between the solutions obtained and the actual optimum solutions by applying an exact solution procedure. However, for problems with more than twenty cities the optimality check could not be conducted considering the inordinate amount of CPU time required for the calculations. Instead, the solutions obtained by using the new algorithms are compared.

In order to apply the algorithms to the MDVRP, the associated cost matrices pass through a transformation so that the TSP solutions to the resultant matrices are the solutions to the MDVRP as well. An obvious result which is of practical importance is that the time required to solve an MDVRP is less than the time required to solve a complete TSP of the same size.

## 1.7 CONTENTS OF THE THESIS

Chapter 2 presents a complete description and formulation of the TSP. A literature survey is made on both the exact and heuristic solution methods for solving the TSP. Algorithms representing different solution techniques are presented. As a result, comments are made on the efficiency and computation effort of different solution procedures.

In Chapter 3, four new algorithms representing distinct heuristic techniques are presented. Each algorithm is described explicitly and used to solve randomly generated examples. For purposes of defining the power of each of these methods, computational results regarding both the efficiency and computation effort are given.

It is well known that the TSP is a subproblem of many other problems frequently encountered in practice. Among them are the VRPs which can be considered as extensions of the TSP. In fact, models representing some of the VRPs are usually built on the TSP as the core model. As a consequence of this fact, some efficient transformations have made it possible to reduce some of the VRPs to a TSP. One of such problems is the MDVRP.

Chapter 4 gives full description and formulation of the MDVRP. The transformation is made more explicit by the use of an example. In addition, one to one correspondence between the TSP and the MDVRP is shown.

Chapter 5 is a treatment of the application of the proposed algorithms to the MDVRP. Each algorithm is used to solve the same example problem in order to make comparisons on the behaviour of the methods.

The concluding chapter in the thesis, Chapter 6, summarizes general conclusions and gives an insight to the extensions of the study which may be subject to further work.

# II. THE TRAVELLING SALESMAN PROBLEM (TSP): A LITERATURE SURVEY

## 2.1 STATEMENT OF THE PROBLEM

Consider a graph G = (N,E) where N = {1,...,n} is a set of n nodes/cities which are to be visited by a salesman and E is a set of arcs/roads joining the nodes. Let $C_{ij}$ be the cost associated with arc (i,j). The problem of finding a tour that includes each node in the graph at least once is known as the General Travelling Salesman Problem (GTSP). The problem of finding a Hamiltonian circuit, a circuit that passes through each node exactly once, with the least cost is the well known TSP.

In this chapter, we review some of the exact and approximate solution methods that have been suggested for solving the TSP. The solution techniques are mainly based on solving the TSP rather than solving the GTSP. Usually, the optimum solution to the TSP is also the optimum solution to the GTSP. However, it follows that if a graph G does not satisfy the triangle inequality then the optimum solution to the TSP may not be the optimum solution to the GTSP. In that case, the GTSP can be reduced to the TSP by a suitable transformation.

Consider that a graph G does not satisfy the triangle inequality (i.e. $C_{ij} > C_{ik} + C_{kj}$ for some $k \neq i$, $k \neq j$) and that one needs to obtain the optimum solution to the GTSP defined in G. A GTSP stated in this manner can be reduced to a TSP by the technique of changing each arc cost $C_{ij}$ to the length of the shortest path between i and j. If an arc (i,j) whose cost is lessened as specified above is contained in the optimum solution to the TSP, then the arc is placed by the shortest path from i to j in the optimum solution and hence, the optimum solution to the GTSP is obtained. As a consequence of the fact that the GTSP is reducible to the TSP, solution techniques for only the TSP are needed.

## 2.2 FORMULATION OF THE TSP

Consider the travelling salesman graph G = (N,E). Let

$$x_{ij} = 1 \quad \text{if arc } (i,j) \in E \text{ is in the tour}$$
$$= 0 \quad \text{otherwise}$$

Then, the problem is

$$\text{minimize} \sum_{i=1}^{n} \sum_{j=1}^{n} C_{ij}x_{ij} \tag{2.1}$$

$$\text{s.t.} \sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1,\ldots,n \tag{2.2}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1,\ldots,n \tag{2.3}$$

The solution must form a tour (2.4)

$$x_{ij} \in \{0,1\} \qquad \forall\ i,j \in N \tag{2.5}$$

Constraint set (2.4) can be written in a number of different ways. Three alternates that have been proposed are

$$\sum_{i\in S}\sum_{j\in \bar{S}} x_{ij} \geq 1 \qquad \forall\ S \subseteq N \tag{2.4a}$$

$$\sum_{i\in S}\sum_{j\in S} x_{ij} \leq |S| - 1 \qquad \forall\ S \subseteq N \tag{2.4b}$$

$$\sum_{(i,j)\in \phi_k} x_{ij} \leq |S_k| - 1 \qquad \forall\ S_k \subseteq N,\ \forall\ \phi_k \subseteq S_k \tag{2.4c}$$

where $S \cup \bar{S} = N$ ,

$\phi_k$ is any Hamiltonian circuit in the induced subgraph $G' = (S_k, E_k)$.

Constraint set (2.4a) ensures that there exists at least one arc between two complementary subsets of nodes of N. Constraint set (2.4b) expresses the fact that no subtour through any subsets of N can exist by imposing that arcs belonging to any subset of nodes, S, cannot be greater than $(|S| - 1)$. Equivalently, constraint set (2.4c) expresses the same fact by restricting the existence of a Hamiltonian circuit in all the induced subgraphs.

It should be noted that the formulation given above takes care of both the symmetric TSP, (i.e. $C_{ij} = C_{ji} \quad \forall\ i,j \in N$) and the asymmetric TSP (i.e. $C_{ij} \neq C_{ji}$ for some $i,j \in N$). However, the problem can be formulated in several different ways when G is undirected and therefore the associated cost matrix is symmetric. Two of such formulations are

$$\text{minimize} \quad \sum_{r=1}^{e} C_r x_r \tag{2.6}$$

$$\text{s.t.} \quad \sum_{r=1}^{e} x_r = n \tag{2.7}$$

$$\sum_{r=K} x_r \geq 2 \qquad \forall\ K \equiv (S,\bar{S}),\ S \subseteq N \tag{2.8}$$

$$x_r \in \{0,1\} \qquad r = 1,\ldots,e \tag{2.9}$$

and

$$\text{minimize} \quad \sum_{r=1}^{e} C_r x_r \tag{2.10}$$

$$\text{s.t.} \quad \sum_{r\in K} x_r \geq 1 \qquad \forall\ K \equiv (S,\bar{S}),\ S \subseteq N \tag{2.11}$$

$$\sum_{r\in E_i} x_r = 2 \qquad i = 1,\ldots,n \tag{2.12}$$

$$x_r \in \{0,1\} \qquad r = 1,\ldots,e \tag{2.13}$$

where e is the total number of arcs in G

$K \equiv (S,\bar{S})$ is an arc cut-set of G which contains arcs (i,j) with $i \in S$ and $j \in \bar{S}$

$E_i$ is the set of arcs incident an node i

$C_r$ is the cost associated with arc r.

Both of the formulations are equivalent since constraints (2.7), (2.8) imply constraints (2.11), (2.12).

## 2.3 SOLUTION PROCEDURES FOR THE TSP

Many solution techniques are available for the TSP [12,13,14].

All of these techniques fall into one of two categories:

a) Techniques that are certain to find an optimum solution but at worst require an inordinate number of calculations (exact solution methods)

b) Techniques that are not always certain to find an optimum solution but require a small number of calculations and therefore less computation effort (heuristic methods).

Exact solution techniques are mainly based on using the advanced results of integer programming, linear programming and dynamic programming as well as enumerating all the existing Hamiltonian circuits of a graph. On the other hand, heuristic algorithms rely upon tour constructing node inserting and node and arc exchanging techniques. In the following sections, we will describe these techniques separately and present algorithms which utilize these methods. Throughout the discussion, the themes touched are related upon to papers in literature. Actually, the aim is not to survey the whole field in the area. Rather, the goal is to give an insight to the techniques existing in literature.

### 2.3.1 Enumeration Methods

In principle, the optimal solution to the TSP can always be obtained by finding all the existing Hamiltonian circuits, calculating their lengths and thus determining the one that is optimal. However, considering complete graphs, the complete enumaration of all the tours

becomes a computationally exhaustive task even for comparatively small size problems. On the other hand, including some simple tests in the computation procedure, the set of all possible tours can be greatly reduced. Then, partial enumeration can be used to select the best tour without considering the excluded ones.

A possible case is that a graph may not contain a Hamiltonian circuit. As a matter of fact, one should first try to establish the existence of a Hamiltonian circuit before proceeding to look for the optimum one. Unfortunately, there exists no easy way for deciding whether or not a graph contains a Hamiltonian circuit. The existing necessary or sufficient conditions are not effective for arbitrary graphs encountered in practical situations.

*Necessary conditions for the existence of a Hamiltonian Circuit*

A necessary condition for the existence of a Hamiltonian circuit is that the graph G = (N,E) be strongly connected. In other words, for any two nodes i,j $\epsilon$ N, there must be a path from i to j. Another necessary condition, however, is that the subgraph, $G_k$, obtained by removing any node k from G, must be unilaterally connected. That is, for any two nodes i,j $\epsilon$ N - {k} in the subgraph there must be a path either from i to j or from j to i. Note that both conditions are necessary but not sufficient for a directed graph to possess a Hamiltonian circuit.

*Sufficient conditions for the existence of a Hamiltonian circuit*

If in a strongly connected directed graph G = (N,E), the degree of each node is greater than or equal to n, where the degree of a node

is the sum of all arcs entering or emanating from that node, then the graph possesses a Hamiltonian circuit.

In the case of an undirected graph the degree of a node is given by the number of arcs incident to that node. In light of this definition the following result due to Chavatal [16] describes a sufficient condition for the existence of a Hamiltonian circuit in an undirected graph:

Let the nodes of an undirected graph G = (N,E) be numbered in such a way that $d(1) \leq d(2) \leq \ldots \leq d(n)$ where d(.) denotes the degree of node (.). For $n \geq 3$ if $d(k) \leq k$, $\forall\, k < n/2$ or equivalently if $d(n-k) \geq n-k$ $\forall k < n/2$, then the graph contains a Hamiltonian circuit. Note that $d(k) \leq k$, $\forall k < n/2$ implies that $d(n-k) \geq n-k$ $\forall k < n/2$.

Actually, it is easy to verify the latter condition. The nodes are first ranked in ascending order of their degrees. Then, the condition is checked for the first (n/2) nodes. Nevertheless, these criteria are too loose to be of value for graphs frequently encountered in practice since they imply the existence of nodes with high degrees. Once these conditions are not satisfied, the only way of determining whether or not the graph contains a Hamiltonian circuit is to make a complete search on the graph. In the following sections we will describe a few algorithms which can successfully be used to find all the Hamiltonian circuits of a directed graph. However, one should keep in mind that even the most efficient algorithm is unable to handle problems with more than twenty nodes with degrees greater than four in a reasonable number of calculations [17].

2.3.1.1 Latin Multiplication Method

The Latin multiplication method enumerates simple paths of lengths 1 through (n-1) in a directed graph G = (N,E). Once all the simple paths of length (n-1) are identified the paths can be completed to Hamiltonian circuits by adding an arc that joins their two end nodes. Then, the least cost Hamiltonian circuit is the optimal solution to the TSP. The algorithm due to Kaufman [18] can be outlined as follows:

1. Define an (nxn) matrix $V^1$ using the cost matrix C in the following way:

   a) Let each entry of $V^1$ be denoted by strings

   b) If $C_{ij} > 0$ $i \neq j$ put $v_i v_j$ in the (i,j) location in $V^1$. Otherwise, put 0 for nonexistent arcs.

2. Define an (nxn) matrix $L^1$. $L^k$ is obtained from $V^k$ by deleting the first node in each nonzero string of $V^k$.

3. Find $V^k \circledast L^i = V^{k+i}$ where $\circledast$ stands for a symbol of Latin multiplication. Latin multiplication is performed like ordinary matrix multiplication as follows:

   a) Zero multiplied by any string is zero.

   b) String multiplications are done by joining two strings into one string, i.e.

   $$V_1V_2V_3 \times V_4V_5V_7 = V_1V_2V_3V_4V_5V_7$$

   c) String additions are written one below the other, i.e.

   $$V_1V_2V_3 + V_4V_5V_7 = \begin{matrix} V_1V_2V_3 \\ V_4V_5V_7 \end{matrix}$$

d) Any string that has a node more than once equals zero.

4. The entries in matrix $V^k$ give the simple paths of length k. $V^{n-1}$ gives the Hamiltonian paths. For all entries representing Hamiltonian paths, check if there exists an arc which connects the terminal nodes of the path. Out of those paths which can be completed to Hamiltonian circuits, choose the one with the least total cost.

Considering the time and storage requirements of the method, even the best computer language cannot provide any advantages on the exhausting need of memory space for finding all the Hamiltonian circuits of comparatively small size problems. However, for problems of less than 20 nodes and an average node degree of less than 3 the algorithm provides a successive means of finding the existing Hamiltonian circuits. In case the graph does not have a Hamiltonian circuit or even a Hamiltonian path, the algorithm can be used to determine all the simple paths upto and including the simple path with the highest cardinality of nodes.

2.3.1.2 Algebraic Methods

In addition to the algorithm presented above the method based on the work of Yau [19], Danielson [20], and Dhawan [21] also uses matrix multiplications to generate all of the simple paths of a graph. The steps of such algorithms are mainly as follows:

1. Let A be a modified adjacency matrix where $a_{rj} = j$ if there

is an arc from r to j. Let $B^k$ be an (nxn) matrix where $b^k_{rj}$ is the sum of the internal node products of all the elementary paths of cardinality k between nodes r and j. The internal node product of a path $i_1, i_2, \ldots, i_k$ is defined as the sequence of nodes $i_2, i_3, \ldots, i_{k-1}$ excluding the two end nodes $i_1$ and $i_k$. Let $B^1$ be the adjacency matrix.

2. Using the ordinary algebraic matrix multiplication obtain the product $B^{k+1} = A.B^k$ where

$$b^{k+1'}_{rj} = \sum_s a_{rs}.b^k_{sj}$$

is the sum of all inner products of all paths from i to j.

3. Obtain $B^{k+1}$ from $B^{k+1'}$ by setting all of its diagonal elements to 0 and eliminating all terms containing node s from $b^{k+1}_{sj}$. The matrix $B^{k+1}$ is the matrix of all elementary paths of cardinality (k+1).

4. Repeat steps (2) and (3) until the path matrix $B^{n-1}$ is generated. The Hamiltonian circuits can be obtained by adding those arcs of the graph which join the terminal nodes of the paths. Alternatively any diagonal element of the matrix obtained from the product $A.B^{n-1}$ also gives the existing Hamiltonian circuits.

Considering the alternative given in setp (4) we may infer that only $b^{n-1}_{11}$ will suffice for determining all the Hamiltonian circuits in the graph. This can be obtained by multiplying only the first

column of $B^k$ at each iteration. As a result, this reduces both the storage and computation effort by considerable factors. However, even with these modifications, the algorithm is still incapable of handling problems of large sizes.

2.3.1.3 Other Enumeration Methods

The two methods presented in the previous sections attempt to find all Hamiltonian circuits at once. As a result, all paths that might take part in forming such circuits have to be stored. Thus, an undesirable increase in the storage requirements results. Contrary to this approach, other enumerative methods consider one path at a time. The path is tried to be extended to form a Hamiltonian circuit. If the path does not lead to a Hamiltonian circuit, then it is modified in such a way that all the possibilities are exhausted. Consequently, the Hamiltonian circuits are found one at a time.

The following enumerative method was first exploited by Roberts and Flores [22]. The steps of the algorithm are as follows:

1. Form a (kxn) matrix D where the entry $d_{rj}$ represents the end node of the $r^{th}$ arc that emanates from node j. Note that the number of rows k of the matrix D corresponds to the largest outdegree of the nodes in the graph G = (N,E). Let $i_1$ be the initial node of path S.

2. Add the first feasible node in column $i_1$ to S. A feasible node is a node that has not already been added to S. If no feasible node can be found then go to step (4). Otherwise, repeat this step until a path of cardinality (n-1) is formed.

3. At this stage, let $S = \{i_1, i_2, \ldots, i_n\}$ where $i_1, i_2, \ldots, i_n$ denotes the sequence by which the nodes appear on path S. If arc $(i_n, i_1)$ exists in G, then a Hamiltonian circuit is found. Find the cost of this circuit and store it if the cost is less than the cost of the circuit that has already been stored in the memory.

4. Remove the last entered node from S. If this removal causes $S = \{\phi\}$ then terminate the algorithm. The Hamiltonian circuit stored in the memory is the optimum solution to the TSP. If no Hamiltonian circuit has been stored, then the TSP is infeasible. If there is at least one node in S, then return to step (2).

Improvements to this method are possible by means of applying a better selection rule for adding the remaining nodes to S. Suppose that at some stage of the algorithm we are searching for a feasible node in column $i_p$ of the matrix D. If there exists a node r in column $i_p$ such that $r \notin S$ and $R^{-1}(r) \subset S$ where $R^{-1}(.)$ is the set of all nodes reaching node (.), then r is the only node that can be added to S since the addition of any other node will exclude r from further consideration and therefore result in a path that cannot lead to a Hamiltonian circuit. On the other hand, if there exists a node r in column $i_p$ such that $r \notin S$, $r \notin R^{-1}(i_1)$ and $R(r) \subset S \cup \{q\}$ for some other node q in column $i_k$ where R(.) is the node set reachible by (.), then q cannot be added to S since the addition of q to S will cause the remaining subgraph not to contain a path from r to $i_1$ and therefore result in a path that cannot lead to a Hamiltonian circuit.

Computational results reveal that, although the tests for the cases mentioned above slow down the procedure for small graphs (less than 20 nodes) they cause a considerable improvement in the computational effort especially for larger graphs.

The method suggested by Roberts and Flores can further be improved by considering the fact that a path constructed in S implies the existence of other paths in the graph. These paths may possibly help to complete a Hamiltonian circuit more quickly or point out that a path S cannot lead to a Hamiltonian circuit. The following algorithm is based on the enumarative scheme proposed by Roberts and Flores and incorporates the improvements developed by Selby [23] and Christofides [24]. The algorithm is summarized in six steps.

0. Let $d^-(j)$ and $d^+(j)$ denote the indegree and outdegree of node j respectively. Select the root node $i_1$ of S as the node with the maximum indegree. Ties are broken by choosing $i_1$ with minimum outdegree. Set $I = \{\phi\}$, $k = 1$, $S = \{i_1\}$ where I is the set of implied arcs and k is the level of the decision scheme.

1. Search for implied arcs, i.e. arcs (j,r) such that $d^-(r) = 1$ or $d^+(j) = 1$. For any such arc (j,r) form the longest path by using (j,r) and all the arcs in I.

   a) If the cardinality of the path is less than (n-1), then add (j,r) to I and remove all the arcs emanating from j and terminating at r. If this removal causes any node q to have $d^-(q) = 0$ or $d^+(q) = 0$ then go to step (5).

b) If the cardinality of the path is (n-1), then check whether the arc $(i_n,i_1)$ exists. If $(i_n,i_1)$ exists, then a Hamiltonian circuit is found, go to step (4). Otherwise go to step (5).

Iterate step (1) until no further arc can be added to I.

2. Check if an implied arc emanates from node $i_k$ say $(i_k,r) \in I$. If $r \equiv i_1$ and $k < n$, go to step (5). Otherwise, set $k = k+1$, $i_k = r$ and $S = S + \{i_k\}$. If $k = n$, then check whether arc $(i_n,i_1)$ exists. If so, a Hamiltonian circuit is found, go to step (4). Otherwise, go to step (5). Iterate step (2) until no further implied arc can be added to S.

3. Select the next node r to be added to S from the nodes not included in S so that r is the node whose $\min\{d^-(r), d^+(r)\}$ is a minimum among all other nodes. Ties are broken by choosing q with $\min\{d^-(r) + d^+(r)\}$. If no feasible node exists then go to step (5). Otherwise, remove all the arcs emanating from $i_k$ and the arcs terminating at r as well as the arc $(r,i_1)$ from the graph. Set $k = k+1$, $i_k = r$. If the removal of the arcs cause any node q to have $d^-(q) = 0$ or $d^+(q) = 0$, go to step (5). If not, return to step (1).

4. Check if the prescribed number of Hamiltonian circuits have been found. If so, terminate the search. Otherwise continue.

5. (a) If k = 1 then stop. All possibilities have been exhausted. Terminate the search. Otherwise, remove node $i_k$ from S.

   (b) If arc $(i_{k-1}, i_k) \in I$, then set k = k-1 and return to step (5a). Otherwise, continue.

   (c) If arc $(i_{k-1}, i_k)$ was added to S at step (3), then reinsert all the arcs removed from the graph at level k, remove all the arcs inserted in I at level k and set k = k-1. Return to step (3).

The algorithm was tested on randomly generated graphs with both the indegree and the outdegree of each node lying in prefixed ranges. As it has been indicated by Martello [25] for node degrees in range 1-3, the algorithm is very fast since a few or no Hamiltonian circuits exist. In case the node degrees range between 2 and 3 the computational effort shows an increase which is proportional to the number of nodes n. Finally, it has been observed that the running times tend to be impractical for node degrees ranging between 2 and 4 and higher.

### 2.3.2 Exact Solution Methods with Branch and Bound

The branch and bound algorithm comprises a theoretical framework for solving different types of combinatorial optimization problems. The method examines successively subsets of the set of all solutions until one of the solutions located in one of the subsets is proven to be optimal.

The set of all solutions is partitioned into a finite number of equivalence classes by using partitioning properties. Then, each

class is examined by using a decision tree. The tree consists of nodes and edges which join the nodes. A path from any node to the root of the tree is called a branch and the solutions are given by the unique branches down the tree.

For each node on the tree, we first check the feasibility of the corresponding solution class. If the solution class does not contain any feasible solution or if the node is terminal, i.e. the solution class cannot be partitioned again, then that node is fathomed (closed). Otherwise an upper bound is calculated and a parameter, which is a numerical value of a special function called the branching function, is defined. This value gives a measure of the desirability for exploring further that particular branch of the search tree. The branching strategies are given different names which vary with the specified branching function. The commonly known strategies include the breadth first strategy, the branch search strategy and the branch and bound strategy.

Termination occurs when either all nodes are fathomed or when all the upper bounds of the unfathomed nodes are less than or equal to the lower bound corresponding to the best feasible solution found so far. If the algorithm does not terminate, then the branching node is selected to be the node having the highest value of the branching function. The new solution class to be examined is obtained by applying a partitioning property given by a special rule called the partitioning rule. Besides, a priority rule is used to determine the subclass to be examined.

In view of the facts mentioned above, a general branch and bound algorithm can be summarized as follows [26]:

1. Let the whole set of solutions be assigned to the root node of the decision search tree.

2. Check for a feasible solution. If the solution class does not contain any feasible solution, then fathom the node and go to step (3). Otherwise,

   a) compute an upper bound for the solution class,

   b) compute a lower bound, if possible,

   c) evaluate the branching function.

3. Terminate the search if either all nodes are fathomed or all the upper bounds of the unfathomed nodes are less than or equal to the current lower bound of the problem. Otherwise, continue.

4. Select the branching node. Use the partitioning rule and the priority rule to determine the new node to be examined. Close the branching node after all the nodes corresponding to the subclasses have been generated. Go to step (2).

Analysing the general branch and bound algorithm, we see that we need a mechanism for finding a feasible solution, a mechanism for computing upper bounds, a termination test, a branching function, a definition of the partitioning properties, a partitioning rule and a priority rule to apply the method properly. Generally, all of the branch and bound algorithms differ depending on the selection of the required information for their application. As a result, there are many branch and bound algorithms which are designed for solving the TSP but which differ in selecting the required information.

Most of the exact solution methods for solving the TSP are of the branch and bound type. At each node of the decision tree, problems which are relaxations of the TSP are solved in order to compute good quality lower bounds. Actually, good quality lower bounds affect the effectiveness of the algorithm much more than any effective branching rules. Therefore, many algorithms found in literature have put emphasis on the problems of calculating lower bounds. The lower bounds are usually calculated from problems which are relaxations of the TSP and whose solution methods are known to be efficient. Among these problems are the assignment problems, the minimal spanning tree problem, matching and covering problems and shortest path problems. The following sections are confined to different branch and bound techniques using these problems for generating lower bounds.

2.3.2.1 The TSP and the Assignment Problems (AP)

Consider the AP and its dual problem defined as follows:

Primal minimize $\sum_{i=1}^{n} \sum_{j=1}^{n} C_{ij}x_{ij}$ (2.14)

s.t. $\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1,\ldots,n$ (2.15)

$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1,\ldots,n$ (2.16)

$x_{ij} \geq 0$ (2.17)

Dual minimize $\sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j$ (2.18)

$$\text{s.t.} \quad u_i + v_j \leq C_{ij} \qquad i,j = 1,\ldots,n \tag{2.19}$$

$$u_i, v_j \quad \text{unrestricted} \tag{2.20}$$

where $C_{ij}$ = cost of arc (i,j) and $C_{ii} = \infty \quad i = 1,\ldots,n$

$x_{ij} = 1$ if arc (i,j) is in the solution set

$= 0$ otherwise.

The AP is a relaxation of the TSP where the additional constraint that the solution must form a tour has been dropped. The AP defined above may have solutions composed of a number of disjoint circuits. One may then impose the additional constraints that have been dropped in order to obtain a single circuit containing all the nodes. The restrictions are usually imposed within the framework of the branch and bound algorithms (Eastman [27], Shapiro [28], Bellmore and Malone [ 29]).

Let the solution of the AP defined above be used in the solution procedure of the TSP. Then, the following branch and bound algorithm can be used to determine the optimum solution to the TSP by imposing the additional constraint that the AP solved on the modified cost matrix gives a single tour.

1. Begin at the live node 0. Solve the AP and let $Z_0$ be the optimal objective function value.

2. Apply the breadth first strategy to select a live node j such that $Z_j = \min_{k \in K} Z_k$ where K is the set of live nodes. If no such j can be found, i.e. all the nodes are fathomed then stop. The problem is infeasible. Otherwise, continue.

3. If the solution in node j is a Hamiltonian circuit, then terminate the algorithm. This is the optimal solution to the TSP. If the solution to node j is composed of a number of independent circuits, then let $\{i_1,i_2,\ldots,i_r,i_1\}$ be the circuit with the minimum number of arcs. Subdivide the problem into r subproblems. In each problem, set the cost of one of the arcs to infinity with all the other costs remaining unchanged. Let each subproblem be represented by the successor nodes of j. Solve the AP for each successor node using the corresponding modified matrix. If a feasible solution to the TSP with an objective function value of $Z_q$ is obtained, then fathom all the nodes whose $Z_k < Z_q$ where $k \in K$. Return to step (2).

Note that the branching rule given above removes the circuit by excluding one of its arcs. However, the subproblems created by using this method are not disjoint. On the other hand, a branching rule which produces mutually exclusive subproblems created by using this method are not disjoint. On the other hand, a branching rule which produces mutually exclusive subproblems is desirable.

The following branching rule can be used in producing disjoint subproblems. Let $\{i_1,i_2,\ldots,i_r,i_1\}$ be the circuit which is going to be removed. Then the cost matrices can be modified as follows:

Problem 1 $\quad C_{i_1 i_2} = \infty$

Problem 2 $\quad C_{i_1 i_2} = -M, \quad C_{i_2 i_3} = \infty$

Problem 3 $\quad C_{i_1 i_2} = -M, \quad C_{i_2 i_3} = -M, \quad C_{i_3 i_4} = \infty$

$\vdots$

Problem r $\quad C_{i_1 i_2} = C_{i_2 i_3} = \cdots\cdots\cdots = C_{i_{r-1} i_r} = -M, \; C_{i_r i_1} = \infty$

-M is a large negative number which ensures that the arc whose cost is assigned -M will remain in the optimal solution. This can also be achieved by deleting the corresponding row and columns of those particular arcs and solving the problem on the reduced matrix.

A better branching rule can be applied to the algorithm by considering the fact that there must be at least one arc leading from the set of nodes that comprise the circuit to the set of the remaining nodes. Each subproblem would be created upon insisting on the existence of such an arc whose initial node is in $S = \{i_1, i_2, \ldots, i_r, i_1\}$ and the final node in $\overline{S} = N-S$. This can be done by setting $C_{jr} = \infty$, $\forall r \in S$ and leaving all other distances unchanged for subproblem j. As a result, the branching rule will lead to the disjoint problems with the following updates:

$$\text{Problem 1} \quad C_{i_1 i_2} = C_{i_1 i_3} = \ldots\ldots = C_{i_4 i_r} = \infty$$

$$\text{Problem 2} \quad C_{i_2 i_1} = C_{i_2 i_3} = \ldots\ldots = C_{i_2 i_r} = \infty$$

$$\text{Problem 3} \quad C_{i_3 i_1} = C_{i_3 i_2} = \ldots\ldots = C_{i_3 i_r} = \infty$$

$$\vdots$$

$$\text{Problem r} \quad C_{i_r i_1} = C_{i_r i_2} = \ldots\ldots = C_{i_r i_{r-1}} = \infty$$

This branching rule and the one presented previously takes only one of the circuits of the solution into consideration. Another branching rule will be one that considers the remaining nodes based on the same reasoning described above.

Suppose that at some node q of the decision tree we have the cost matrix C which represents the graph $G^q = (N, E^q)$. Let $S = \{i_1, \ldots, i_r\}$ be the node set representing a circuit in the solution associated with

node q and $\bar{S}$ = N-S. Then the new branching rule requires the following [29]:

Problem 1 $\quad C_{i_1 j} = \infty \quad \forall j \in S, \quad j \neq i_1$

Problem 2 $\quad C_{i_1 j} = \infty \quad \forall j \in \bar{S}, \quad C_{i_2 j} = \infty \; \forall j \in S, \quad j \neq i_2$

Problem 3 $\quad C_{i_1 j} = C_{i_2 j} = \infty \quad \forall j \in \bar{S}, \quad C_{i_3 j} = \infty \quad \forall j \in S, \; j \neq i_3$

$\vdots$

Problem r $\quad C_{i_1 j} = C_{i_2 j} = \ldots\ldots = C_{i_{r-1} j} = \infty \; \forall j \in \bar{S}, \; C_{i_r j} = \infty$

$\forall j \in S, \quad j \neq i_r$

As it has been described previously an efficient branching strategy would be the breadth first strategy as it is used in the given algorithm. But one could also use the depth first strategy and therefore branch to one of the successor nodes of a node just partitioned. Note that the termination criteria remain the same in both cases.

No matter which branching strategy is used, the quality of the lower bounds computed has a significant influence on the number of branchings in the decision tree and therefore on the computational efficiency of the branch and bound method. The objective function value of the AP is a valid lower bound and can be used quite efficiently. However, a tighter bound can be calculated from the optimal solution to the AP at the expense of a little extra effort [17].

Let the optimal solution to the AP contain $n_1$ disjoint circuits. Then, each circuit is contracted so that all the circuits, $q_i^1$, are represented by single nodes. Construct a graph $G^1 = (S^1, E^1)$ where $S^1 = \{s_1^1, s_2^1, \ldots, s_{n_1}^1\}$ and each node $s_j^1$ represents a circuit $q_j^1$ and $E^1$ is the arc set taken as

$$E^1 = \{(s_i^1, s_j^1) \mid c^1_{s_i^1, s_j^1} = \min_{\substack{r \in q_i^1 \\ k \in q_j^1}} \{c^1_{rk}\}\}$$

where $[c^1_{s_i^1, s_j^1}]$ are the elements of the resultant cost matrix. The AP is solved once more on the contracted problem using matrix $C^1$. However, the solution to this problem may also contain $n_2$ disjoint circuits. Note that these circuits have the previous circuits as nodes. The new circuits, $q_i$, are further contracted into nodes to form a new graph, $G^2 = (S^2, E^2)$ where $S^2 = \{s_1^2, s_2^2, \ldots, s_{n_2}^2\}$ is the set of $n_2$ nodes each representing a circuit $q_i$ having the previous circuits as nodes and $E^2$ is the edge set taken as

$$E^2 = \{(s_i^2, s_j^2) \mid c^2_{s_i^2, s_j^2} = \min_{\substack{r \in q_i^2 \\ k \in q_j^2}} \{c^1_{rk}\}\}$$

The AP solved for this doubly contracted problem may still have $n_3$ disjoint circuits. Thus, the contraction is continued iteratively until the problem is reduced to a single node.

An important point that should be taken care of is that the cost matrix obtained at the end of each iteration must satisfy the triangle inequality. If the cost matrix produced fails to satisfy the triangle inequality, it has to be transformed into one that does. This procedure is called compression and compression is performed by replacing every element $C^k_{s_i^k\, s_j^k}$ for which

$$C^k_{s_i^k, s_j^k} > C^k_{s_i^k\, s_r^k} + C^k_{s_r^k\, s_j^k} \quad \text{for some} \quad s_r^k \in S^k$$

by the value of

$$\min_{s_r^k \in S^k} \{C^k_{s_i^k\, s_r^k} + C^k_{s_r^k\, s_j^k}\}$$

As it has been shown by Christofides [17] the sum of the values of the solutions to the APs obtained during the "solution-contraction-compression process" is a valid lower bound to the TSP. The computation effort for the contraction and compression parts of the process is known to vary in order of n and the time required to calculate the bound is approximately 14.3% greater than the time required to solve an AP of the same size. This increase in time, however, results in a considerable amount of saving in the decision tree search.

Another bound which also uses the AP was introduced by Balas and Christofides [30]. They consider the introduction of some violated constraints of the TSP into the objective function by the use of Lagrange multipliers. In addition to the AP formulation, the TSP includes constraints that forces the solution to form a tour. Such constraints are given by (2.4a), (2.4b) and (2.4c). Let $\lambda_p$ be the multiplier associated with the $r^{th}$ constraint of the ste (2.4a) which is not satisfied. The problem, then, becomes

$$\text{minimize} \quad \sum_i \sum_j C_{ij}x_{ij} - \sum_r \lambda_r \sum_{i\varepsilon S_r} \sum_{j\varepsilon \bar{S}_r} x_{ij} + \sum_r \lambda_r \tag{2.21}$$

$$\text{s.t.} \quad \sum_i x_{ij} = 1 \qquad j = 1,\ldots,n \tag{2.22}$$

$$\sum_j x_{ij} = 1 \qquad i = 1,\ldots,n \tag{2.23}$$

$$x_{ij} \geq 0 \quad \forall i,j \in N \tag{2.24}$$

An approximate method for finding these multipliers can be given as follows:

1. Form the graph $G_o = (N,E_o)$ where $E_o = \{(i,j):\overline{C}_{ij} = 0\}$ and $[\overline{C}_{ij}]$ are the elements of the cost matrix after the AP is solved.

2. For each node i, find R(i) the node set reachible from i via arcs in $G_o$. If the number of reachible nodes is (n-1) $\forall i$, then stop. The multipliers are calculated. Otherwise, generate cuts for the nodes whose reaching sets are incomplete. Let k be one of such nodes whose reaching set is given by R(k). For $\overline{R}(k) = N-R(k)$ the corresponding Lagrange multiplier $\lambda_k$ is calculated as

$$\lambda_k = \min_{\substack{i\in R(k) \\ j\in\overline{R}(k)}} \{C_{ij}\}$$

and the cost matrix is updated by

$$\overline{C}_{ij} = \overline{C}_{ij} - \lambda_k \qquad \forall i \in R(k), \quad j \in \overline{R}(k)$$

3. Update $G_o$ so that arcs for which $\overline{C}_{ij}$ has become 0 are included in $G_o$. Return to step (2).

At this stage it is probable that there exists some unsatisfied constraints of type (2.4b). Once again, these constraints can be introduced by using a further Lagrangean relaxation. Let $\mu_q$ be the multiplier associated with the $q^{th}$ unsatisfied constraint of the set (2.4b). The new problem can be stated as

$$\text{minimize} \quad \sum_i \sum_j C_{ij}x_{ij} - \sum_r \lambda_r \sum_{i\in S_r} \sum_{j\in\overline{S}_r} x_{ij} + \sum_r \lambda_r$$
$$+ \sum_q \mu_q \sum_{i,j\in S_q} x_{ij} - \sum_q \mu_q |S_q| + \sum_q \mu_q \qquad (2.25)$$

$$\text{s.t.} \quad \sum_i x_{ij} = 1 \qquad j = 1,\ldots,n \tag{2.26}$$

$$\sum_j x_{ij} = 1 \qquad i = 1,\ldots,n \tag{2.27}$$

$$x_{ij} \geq 0 \qquad \forall i,j \in N \tag{2.28}$$

Then, the procedure continues with the following step.

4. Calculate $\mu_q$ in a similar way that the duals are computed for the AP. For each $\mu_q$ determined, calculate new dual variables $u_i$ and $v_j$ for the AP and update the costs by $\bar{C}_{ij} = \bar{C}_{ij} - u_i - v_j$.

Note that at the end of this procedure, the initial AP solution is still optimal and

$$TC_{AP} + \sum_r \lambda_r + \sum_q \mu_q$$

with $TC_{AP}$ being the optimal objective function value of the AP, constitutes a lower bound to the TSP.

Since the number of zero elements of the cost matrix has been increased after the application of step (1)-(4) mentioned above, it might be the case that those elements comprise a Hamiltonian circuit although we know that the initial AP solution is still optimal. If there is not any Hamiltonian circuits then the lower bound can be improved by applying the following step.

5. Consider the final version of $G_0$. Let $G_0^s = (N^s, E_0^s)$ be the graph generated from $G_0$ by removing node s. If $G_0^s$ is not unilaterally connected then there must be a pair of cuts $K_s^1$, $K_s^2$ of $G_0^s$ for which $E_0^s \cap K_s^1 = E_0^s \cap K_s^2 = \phi$. Define

$$\pi_s = \min_{(i,j)\in K_s^1 \cup K_s^2} \{\bar{C}_{ij}\}$$

and make the transformation

$$\bar{C}_{ij} = \bar{C}_{ij} - \pi_s \qquad \forall (i,j) \in K_s \cup K_s$$

Perform this procedure for every $s \in N$ so that $G_o^s$ is unilaterally connected after the removal of any node s.

As a result, the quantity

$$TC_{AP} \quad \sum_r \lambda_r + \sum_q \mu_q + \sum_s \pi_s$$

is a valid lower bound for the TSP. Note that the procedure can be applied to both symmetric and asymmetric problems but produces better bounds for asymmetric problems [31].

After all, the use of the AP based bounds have been observed to perform well in tree search algorithms. Up to date results reveal that problems with 250 or more nodes can successfully be solved by the use of the AP based bounds with different types of branching schemes [30].

2.3.2.2 The TSP and Minimal Spanning Tree Problems

The minimal spanning tree problem (MSTP) is the problem of finding the tree that spans all the nodes of a graph with the minimum total cost. The TSP is closely related to the MSTP in the sense that the problem of finding the shortest Hamiltonian path of a graph is equivalent to the problem of finding the minimal spanning tree of a graph with the additional constraint that no node should have a degree

greater than 2. The minimal spanning tree of a graph may contain arcs which result in a degree $d(i) > 2$ for some node i. If such a node i exists, then at least one of the arcs incident to node i must be eliminated. Thus, there are d(i) problems which must be taken into consideration. In each one of the d(i) problems, one of the arcs incident to node i is eliminated ($C_{ij} = \infty$) and the MSTP is solved again in order to see if the absence of the eliminated arc leads to a Hamiltonian path.

The following branch and bound algorithm can be used to determine the shortest Hamiltonian path with the aid of the MSTP [17]:

1. Begin at the live vertex 0. Solve the MSTP. Let $Z_o$ be the cost of the minimal spanning tree.
2. Find a live node j on the decision tree such that $Z_j$ is minimum. If no live node can be found then stop, the problem is infeasible. Otherwise, continue.
3. If the solution to node j is a Hamiltonian path then stop, the optimum solution is obtained. Otherwise, select a node i on the spanning tree such that $d(i) > 2$. Subdivide the problem into d(i) subproblems. In each problem, set the cost of one of the arcs incident to i to infinity, while all other costs remain unchanged. Let each subproblem be represented by the successor nodes of j on the decision tree. Solve the MSTP for each successor node. If a feasible solution (i.e. a Hamiltonian path) is found with an objective function value $Z_q$, then fathom all the live nodes, k, where $Z_k < Z_q$, $k \in L$, and L is the set of all live nodes. Return to step (2).

Note that this algorithm deals with finding the minimal spanning tree of a graph rather than finding the shortest Hamiltonian circuit which is the solution to the original TSP. However, once a solution method for finding the shortest Hamiltonian path is known, a small modification will suffice for dealing with finding the shortest Hamiltonian circuit.

Let the shortest 1-tree of a graph, G, be defined as the minimal spanning tree of the subgraph of G with node 1 removed, plus the two shortest arcs from node 1 to two other nodes of the tree [14]. Then, the shortest 1-tree with all node degrees of value 2 is the shortest Hamiltonian circuit of the graph. Thus, the branch and bound method discussed above can be used to solve this problem as well.

Instead of using the cost of the shortest spanning tree as a lower bound by itself, one may count the longest branch on the tree twice and then let the overall cost be a better bound to the optimal TSP. This follows from the fact that the spanning tree contains (n-1) arcs that connect all of the n nodes whereas n arcs are needed to comprise a Hamiltonian circuit. Consequently, since the longest arc on the shortest Hamiltonian circuit is at least as long as the longest arc on the minimal spanning tree the quantity

$$TC_{MST} + \max_{(i,j)\in T} C_{ij}$$

where $TC_{MST}$ is the cost of the minimal spanning tree and T is the set of arcs on the minimal spanning tree, is a lower bound to the shortest Hamiltonian circuit.

Another way of deriving a better bound by using the MSTP is to

include the constraints $d(i) \leq 2$, $\forall i \in N$ to the objective function by means of Lagrange multipliers. In other words, nodes with degrees greater than 2 are penalized. There are many penalizing procedures proposed for solving the TSP with the aid of minimal spanning trees. Methods of this type were first exploited by Held and Karp [14] and Christofides [24]. Improved methods for deriving the penalties were later introduced by Hansen and Krarup [32].

The method due to Volgenant and Jonker [33], uses arc exchanges in minimal trees in combination with a branch and bound algorithm based on the 1-tree relaxation. Once a minimal spanning 1-tree, T, is obtained the method distinguishes two types of arcs:

1. Arcs not incident to node 1

   For an arc (i,j) in T, the 1-tree $T_{ij}^{-}$ follows from T by exchanging (i,j) with a shortest arc (r',s') not in T in its fundamental cut set. Now arc (i,j) must be part of an optimal solution if

   $$C_{T_{ij}^{-}} = C_T - C_{ij} + C_{r's'} > u$$

   where $C_{T_{ij}^{-}}$ is the cost of the 1-tree following from the original 1-tree whose cost is given by $C_T$ and u is an upper bound on the optimal TSP value.

   For an arc (k,q) not in T, the 1-tree $T_{kq}^{+}$ follows from T by exchanging (k,q) with a longest arc (r",s") on its fundamental path in T. Now arc (k,q) cannot be part of an optimal solution, if

$$C_{T^+_{kq}} = C_T + C_{kq} - C_{r"s"} > u$$

where $C_{T^+_{kq}}$ is the cost of the 1-tree following from T.

2. Arcs incident to node 1

Let (1,i) and (1,j) be the arcs of the minimal 1-tree T and $C_{1i} < C_{1j}$. Let k be an index with $C_{1k} = \min\{C_{1q} | q \in N, q \neq i,j\}$. Then, the 1-tree $T^-_{1i}$ and $T^-_{1j}$ follow from T by exchanging arcs (1,i) respectively (1,j) with the arc (1,k) which is not in T. So, an arc (1,j) must be part of an optimal solution if

$$C_{T^-_{1j}} = C_T - C_j + C_{1k} > u$$

Similarly, the 1-tree $T^+_{1k}$, $k \in N$, $k \neq i,j$ follows from T by exchanging an arc (1,k) not in T with arc (1,j) in T. Thus, arc (1,k) cannot be part of an optimal solution, if

$$C_{T^+_{1k}} = C_T + C_{1k} - C_{1j} > u$$

The TSP algorithm of Volgenant and Jonkers is based on the 1-tree relaxation of Held and Karp [14,15] and modified with the edge exchanges on one major point: Using a minimal 1-tree in one of the live nodes of the decision tree, the branching is governed by the $C_T$- values of the arcs incident to an arbitrary node i with d(i) > 2 on the subtour of the minimal 1-tree. The set of feasible solutions is split into three subsets. The first set is characterized by requiring to edges, say $e_1$ and $e_2$, incident to i; the second set by forbidding $e_2$ and keeping $e_1$ required and the third set by forbidding $e_1$ only. As $e_1$ and $e_2$ the arcs with the largest respectively second

largest $C_T$- value are chosen. Throughout the algorithm a heuristic subalgorithm is used on simply chained 1-trees to obtain a better upper bound for the TSP, so that more variables can be eliminated and more sensitive $C_T$- values can be calculated.

As it has been reported, computational results has shown that the arc exchanges are advantageous for Eucledian problems up to 120 nodes as well as for random table problems upto 200 nodes. Nevertheless, up to date results reveal that problems up to 100 nodes can be solved successfully be embedding the MSTPs as lower bounds into branch and bound algorithms.

2.3.2.3 The TSP and Matching Problems

This section presents a method for calculating a lower bound on the length of an optimum Hamiltonian circuit by the use of the matching problem. Given an undirected graph $G = (N,E)$ a subset $D \subseteq E$ is called a b-matching of G, if the node degrees $d(i) = b$ for all $i \in N$. Then the problem of finding a minimum cost b-matching is the integer programming problem

$$\text{minimize} \quad \sum_{k\in E} C_k x_k \tag{2.29}$$

$$\text{s.t.} \quad \sum_{k\in A_i} x_k = b \qquad i = 1,\ldots,n \tag{2.30}$$

$$x_k \in \{0,1\} \qquad \forall k \in E \tag{2.31}$$

where $A_i$ is the set of arcs incident to node i. Adding the constraint

$$\sum_{k\in K_r} x_k \geq 1 \qquad \forall\, K_r \equiv (S_r,\overline{S}_r)\ ;\ S_r \subset N \tag{2.32}$$

to the 2-matching problem we obtain the formulation of the symmetric TSP. Thus, the 2-matching problem is a relaxation of the TSP and can therefore be used as a valid lower bound. On the other hand, the additional constraints can be included into the objective function by means of Lagrange multipliers. Hence, we obtain a problem which can be solved by a technique similar to the one used in deriving a lower bound via the assignment problem.

Occasionally, the lower bound obtained by solving a 2-matching problem can be embedded into a decision tree search algorithm. Experiments showed that the lower bounds generated by the 2-matching problem are much better than the lower bounds generated by the assignment problem when the graph is symmetric.

2.3.2.4 The Shortest n-Paths and the TSP

Consider a Hamiltonian circuit. Obviously, this is an n-path from a node j back to j where each node appears once and only once on the path. On the other hand, excluding the restriction that each node must appear exactly once on the path, the computation of the shortest n-path from j back to j becomes a simple problem which can be solved by dynamic programming. Note that a node can appear an arbitrary number of times on the shortest n-path. The recursion formulae for the computation of such a path can be given as

$$f_1(i) = C_{ji} \qquad \forall i \in N, \quad i \neq j \tag{2.33}$$

$$f_k(i) = \min_{\substack{q \in N \\ q \neq i,j}} \{f_{k-1}(q) + C_{qi}\} \qquad i \neq j, \quad i \in N, \quad k = 2,\ldots,n-1 \tag{2.34}$$

$$f_n(j) = \min_{\substack{q \neq j \\ q \in N}} \{f_{n-1}(q) + C_{qj}\} \tag{2.35}$$

Eventually, if the n-path passes through each node exactly once, then it is the solution to the TSP. If a node appears on the path more than once, then $f_n(j)$ can be used as a lower bound on the value of the TSP.

A better bound can be derived by penalizing the nodes which appear on the path more than once. Let the costs $C_{ij}$ be transformed by $\overline{C}_{ij} = C_{ij} + \lambda_i + \lambda_j$ where $\lambda_i$ is a penalty associated with node i. Then the cost of any Hamiltonian circuit in the graph is increased by the same constant amount $2\sum_i \lambda_i$. On the other hand, n-paths that are not Hamiltonian circuits are penalized by first computing $f_n(j)$ with the modified costs $\overline{C}_{ij}$. Let the n-path pass through node i $k_i$ times. Then

$$\omega(\lambda) = f_n(j) + 2 \sum_i (k_i - 1)\lambda_i \tag{2.36}$$

is a valid bound to the TSP. The problem is therefore to choose that $\lambda^*$ which corresponds to the maximum of the expression

$$\omega(\lambda^*) = \max_{\lambda} \{\omega(\lambda)\} \tag{2.37}$$

and use $\omega(\lambda^*)$ as a lower bound for the TSP. Subgradient optimization is one possible procedure for solving this problem.

As it has been reported by Houck et.al [34] one type of node repetition can be prevented by a simple modification on the recursion formulas. In other words, it is possible to exclude occurrences where the $r^{th}$ and $(r+2)^{th}$ nodes on the n-path correspond to the same node

in the graph for some value of r. The quantity calculated by solving the shortest n-path with the modified recursion formulas plus the associated penalties is a better lower bound to the TSP.

An advantage of the lower bounds using the shortest n-paths is that additional constraints for problems related to the TSP can easily be included in the structure of the problem. On the other hand, the fact that $O(n^3)$ operations are required to compute the shortest n-path as compared with $O(n^2)$ operations for the minimal spanning tree problem and $O(n^{2.5})$ operations for the assignment problem is a disadvantage of the method.

2.3.2.5 Little's Branch and Bound Algorithm

The basis of Little's algorithm [11] is to first identify a feasible solution to the TSP and then to decompose the set of all remaining feasible tours into smaller and smaller subsets. At each step of the decomposition, the bounds provide a guide for partitioning the subsets of a feasible tour. A tour with a length less than the length of the current best tour is assigned to be the minimum lower bound of all the tours. The process of bounding tours, eliminating the suboptimal alternatives and branching continues until all of the bounds on the decision search tree are greater than or equal to the length of the best available tour.

The algorithm starts with the original cost matrix C and subtracts from every entry in each row the minimum element of that row and repeats this process for all the rows. Then, the minimum element of each column is subtracted from every entry in that column in the resultant cost matrix. The process of subtracting the minimum element

from the entries in each row and column is called row reduction and column reduction respectively. The reduced matrix contains at least one zero in each row and in each column. Since all the elements in the reduced matrix are nonnegative, the sum of the reduced constants, H, constitutes a lower bound on the length of any tour under the matrix before reduction.

The next step is to identify the minimal length tour by assigning one zero valued cell in each row and column. If such a zero valued tour can be found, then this is the optimal solution. However, the arcs of the optimal tour are not identified simultaneously. The tour is formed by selecting one arc at a time from the cost matrix.

As it has been suggested by Little, a penalty is calculated for each zero element in the cost matrix. The penalties, $\bar{p}_{ij}$, give the minimum cost that would be incurred if the optimum tour does not contain the arc (i,j). Thus, that arc whose cost under the reduced matrix is zero and whose penalty is the largest governs the partitioning of the solution set. The total number of tours is divided into two subsets; those that include arc (i,j) and those that do not. Let these subsets be represented by two subsequent nodes on the decision tree. The bound on the node which represents the tours not including arc (i,j) is $(H+\bar{p}_{ij})$. Before we can determine the new bound on the node which represents the tours that include arc (i,j), certain modifications have to be performed in the cost matrix. Since arc (i,j) is selected to appear in the final tour it is impossible to include another arc corresponding to an entry in row i or column j. Thus, row i and column j are deleted from the cost matrix. Finally, costs of arcs which if not taken

out of consideration might create subtours are set to infinity. After these modifications are made the cost matrix is further reduced so that each row and column contains at least one zero. The bound on the node is now computed as the sum of the new reducing constants plus the lower bound of the predecessor node. As a result, a new branching becomes possible. The subset of all tours is partitioned into smaller subsets. The partitioning process continues until the final subset contains a single tour. Furthermore, the branching process is controlled by the lower bounds. The subset of tours whose lower bound is larger than the lower bound of a node representing the best feasible tour are deleted from further consideration. That is, no additional branching is performed from the corresponding node. The algorithm is summarized in the following steps:

1. Begin at the live node 0. Let $Z_0 = 0$.
2. Reduce C. Set H to the sum of reducing constants. Set the lower bound of the node to the sum of H plus the lower bound of the predecessor node. If the lower bound is greater than the cost of the best tour available, go to step (6).
3. Calculate the penalty for each zero element in C. Choose arc (q,r) such that $\overline{p}_{qr} = \max_{i,j}\{\overline{p}_{ij}\}$. Set the bound of the node which represents the subset of all tours not including arc (q,r) to the lower bound of the predecessor node plus the penalty $\overline{p}_{qr}$.
4. Branch to the node which represents the subset of tours that include arc (q,r). Cross out row q and column r.

Insert infinities in C to prevent subtours from being formed. If C is not a (2x2) matrix, then return to step (2). Otherwise, continue.

5. Since C is now a (2x2) matrix, a tour has been obtained. If the cost of this tour is less than the cost of the best available tour, then record it. Otherwise, continue.

6. Select the next node to branch from, as the node with the least lower bound. If all the bounds are greater than the least cost tour, then stop. The tour stored is optimal. Otherwise continue.

7. Update and set up matrix C so that it corresponds to the node selected in step (6). Return to step (3).

Little's method has many advantages as compared with other branch and bound techniques. The method can be extended to handle additional constraints which are not included in the TSP, but may appear in problems which are closely related to the TSP. Another important property is that if for any reason the tree search is stopped before the search ends with an optimal solution, then a good and sometimes the optimal solution is obtained. But, similar to all other branch and bound methods the computational complexity of this method is exponentially dependent on the number of nodes of the problem. In other words, the combinatorial structure of the TSP is still in effect.

### 2.3.3 Dynamic Programming Solution of the TSP

An alternative solution to the TSP by means of dynamic programming has been offered independently by Bellman [35], Held and Karp [36] and Gonzales [37]. The procedure is more general than the branch and bound technique and requires less computation effort. However, the storage requirements for dynamic programming are more limiting as compared with the branch and bound technique.

Consider the n-node TSP with costs specified by the elements of matrix C. Let node 1 be the origin of the travelling salesman tour. Considering that i is any node other than node 1, define the following:

$S_k$ = a set of k nodes other than nodes 1 and i

$\overline{S}_k$ = a set consisting of the remaining (n-k-2) nodes

Suppose that starting at node 1 on the optimal tour, a path passes through each of the nodes of $\overline{S}_k$ in some particular order and ends at node 1. Note that the nodes in $S_k$ have to be included in the path in some order before returning to node 1. That is, the portion of the tour from node i through the nodes of $S_k$ and back to node 1 has to be considered. Obviously, this will be the shortest possible path from node i back to node 1 passing through k nodes of $S_k$. Let $f(i,S_k)$ be the shortest possible path from i back to 1 with k nodes of $S_k$ in between. Then, the recursion formulas can be given as

$$f(i,\phi) = C_{1i} \qquad k = 0 \qquad (2.38)$$

$$f(i,S_k) = \min_{j \in S_k} \{C_{ij} + f(j,S_k - \{j\})} \qquad k = 1,\ldots,n-1 \qquad (2.39)$$

Note that $f(1,S_{n-1})$ would be the length of the optimal tour of the TSP.

The formulation of the TSP by dynamic programming can be simplified for the case of symmetric cost matrices. If the total number of nodes of such a problem is n, then this number can be expressed as (2q+1) if it is odd and as 2q if it is even. If n is odd, the recursion formulas given above can be used recursively from k = 0 to k = q to obtain an optimal path of length (q+1). On the other hand, since the cost matrix is symmetric, the path including the remaining nodes would have already been computed and hence the problem is solved. If n is even, then the procedure remains the same except that the recursion ranges from k = 0 to k = q-1.

As it can be seen, the storage requirements for the problem is extremely large. One must be able to store all the computations at two consecutive stages since it is not possible to overwrite any of the computations made at a given stage until all the computations at the following stage have been made. As a concluding remark, we can state that the storage requirements for dynamic programming are more than doubled for each additional node. Unfortunately, even the best methods developed are not able to overcome this difficulty.

### 2.3.4 Exact Solution Methods Based on Linear Programming

The TSP cannot be directly formulated and solved as a linear programming problem in practice. However, a possible procedure for solving the TSP is to solve its relaxations by means of linear programming and then to impose the relaxed constraints by either a branch and bound algorithm or a cutting plane procedure. Actually, there

constraints would be taken into consideration when they are violated by the linear programming solution of the relaxed problem.

The basic method described as above has been adopted in numerous different ways. For example, consider the method proposed by Crowder and Padberg [38] for the symmetric TSP. The basic idea of applying their method goes as follows: First the linear program

$$\min\{Cx \mid Ax = 2, \quad 0 \leq x \leq 1\} \tag{2.40}$$

where C is the vector with (n(n-1)/2) components given by the arc distances and A is the incidence matrix of the complete graph is started with a feasible solution. If the next feasible solution is a tour, a usual pivot is carried out. Otherwise, the next feasible solution is chopped off by some cutting plane which is satisfied by the current solution at equality. Consider the constraints

$$\sum_{i=0}^{k} x(S_i) \leq |S_0| + \sum_{i=1}^{k} (|S_i| - 1) - \left\lceil \frac{1}{2} k \right\rceil \tag{2.41}$$

where $\lceil . \rceil$ denotes the next highest integer, $|S|$ denotes the cardinality of set S and the sets $S_i$ are proper subsets of N satisfying the following conditions for i = 0,1,...,k.

$$|S_0 \cap S_i| \geq 1 \quad i = 1,\ldots,k \tag{2.42}$$

$$|S_i - S_0| \geq 1 \quad i = 1,\ldots,k \tag{2.43}$$

$$|S_i \cap S_j| = 0 \quad 1 \leq i \leq j \leq k \tag{2.44}$$

$$k \text{ odd} \tag{2.45}$$

The arc set $\{U_{i=0}^{k} E(S_i)\}$ is called a comb in G and the inequalities (2.41) are called comb constraints. A comb with k = 1 and $|S_0| = 1$ is a subtour elimination constraint [39]. A comb is a 2-matching constraint [40] if the inequalities both (2.42) and (2.43) hold as equalities. As a result, in order to introduce a cutting plane into the linear program, some suitable subtour elimination, 2-matching and comb constraints are identified by the use of (2.41). Once a usual pivot is executed on the enlarged linear program, a tighter relaxation of the TSP is obtained. Continuing in this manner, a situation where no suitable constraint can be found is encountered. Then, the next step is to reduce the problem under consideration in size by fixing variables at either zero or one utilizing the fact that both a value for a tour which is obtained by applying a heuristic due to Lin and Kernighan and a true lower bound on the optimum tour length (i.e. the current solution of the LP) have been obtained. Let $\bar{c}_j$ be the reduced cost of the corresponding optimal tableau and $\Delta$ denote the difference between the cost of the best tour obtained so far and the optimum value of the objective function of the linear program. Then, all nonbasic variables whose $\bar{c}_j \geq \Delta$ in the optimal tableau are fixed with value zero and all nonbasic variables whose $-\bar{c}_j \geq \Delta$ in the optimal tableau are fixed with value one. Thus the linear program is reduced in size and takes the form

$$\min\{C^R y | A^R y = b \ , \quad Dy \leq d \ , \quad 0 \leq y \leq 1\} \tag{2.46}$$

where $C^R$ are the costs of the arcs whose corresponding variables could not be fixed at either zero or one, $A^R$ is the corresponding node-arc

incidence matrix, b is a vector with components equal to 0, 1 or 2, D is a matrix corresponding to the cutting planes generated and d is the corresponding right handside adjusted for the variables fixed at value one.

Once the linear program (2.46) is solved, a branch and bound procedure is used to find an optimal zero-one solution. If the optimal solution defines a tour, then the optimal solution to the TSP is found. If the zero-one solution defines a collection of subtours in the graph then the subtour elimination constraints are appended to the program. This new linear program is reoptimized starting with the optimal basis from the previously solved linear program. Then, the branch and bound technique is used again and the procedure is iterated. After finitely many steps the procedure finds the minimum length tour of the graph.

Other algorithms using the same basic idea have been proposed by Miliotis [41], Grötschel [42], and Christofides and Whitlock [43]. We will not go into the details of these algorithms but instead state a general result on their performance. Comparing the linear programming based method with pure branch and bound procedures, we see that they are competitive with branch and bound methods for solving symmetric TSPs whereas they are not competitive for asymmetric cases [31].

### 2.3.5 Approximate Methods for the TSP

In this section, we analyse techniques that are not always certain but find a near optimum and sometimes the optimum solution

to the TSP with a reasonable number of calculations. The algorithms corresponding to these approximation methods have been observed to run faster than the best known exact solution methods. In view of the computational difficulties that arise from the exponential computation time dependent on the number of nodes, the approximate algorithms become preferable although they may not produce an optimal tour. Furthermore, some of these methods have known bounding ratios of the obtained total cost to the optimal tour cost. The ratios are dependent on the number of nodes in some cases and constant in others.

It is possible to classify the techniques in different categories according to their algorithmic approaches. These categories are:

i) tour building techniques
ii) successive improvement techniques
iii) techniques using minimal spanning trees.

2.3.5.1 Tour Building Techniques

2.3.5.1.1 Insertion Methods

The basic idea of the insertion methods is to start with a partial tour and construct subtours progressively each time with an increase in the number of nodes. That is, each time one node is inserted into the partial tour. Then, the new partial tour is used in the same way to obtain another partial tour. The procedure is continued until all nodes are covered.

2.3.5.1.1.1 Nearest Insertion Method

The first insertion method we study is the nearest insertion method. The corresponding algorithm can be summarized as follows:

Given a graph $G = (N,E)$ and a subtour $T_j = \{i_1, i_2, \ldots, i_j, i_1\}$ with cardinality $j$, construct another subtour $T_{j+1}$ by performing the steps described below.

a) Find a node $i_k \in T_j$ such that

$$C_{i_k,r} = \min_{q \in N-\{T_j\}} \{\min_{s=1,..,j} \{C_{i_s,r}\}\}$$

where C is a symmetric cost matrix satisfying triangular inequality.

b) Delete arc $(i_k, i_{k+1})$ in $T_j$ and add arcs $(i_k, r)$ and $(r, i_{k+1})$ to obtain the new subtour $T_{j+1}$ and let the new sequence of nodes be $\{i_1, i_2, \ldots, i_j, i_{j+1}\}$. Note that $i_{k+1} = i_1$ if $k = j$.

c) Repeat steps (a) and (b) until $T_n$ is obtained.

It has been proven that the ratio of the tour cost obtained by the nearest insertion method to the optimal tour cost is less than 2 [44]. As a result, this method can be programmed to run in polynomially bounded time with an order of $n^2$ where n stands for the total number of nodes.

2.3.5.1.1.2 The Cheapest Insertion Method

Similar to the nearest insertion method the cheapest insertion method produces a tour no worse than twice the optimal regardless of the number of nodes. The algorithm can be outlined as follows:

Given a graph G = (N,E) and a partial tour $T_j = \{i_1,\ldots,i_j,i_1\}$ of cardinality j, construct another subtour $T_{j+1}$ by performing the following steps.

a) Find a node q such that

$$TC_{j+1} = \min_{\substack{q\in N-\{T_j\} \\ i_k\in T_j}} \{TC_j + c_{i_k,q} + C_{q,i_{k+1}} - C_{i_k,i_{k+1}}\}$$

where $TC_j$ is the cost of tour j and $i_{k+1} = i_1$ if k = j.

b) Delete arc $(i_k,i_{k+1})$ in $T_j$ and add arcs $(i_k,q)$ and $(q,i_{k+1})$ to obtain the new subtour $T_{j+1} = \{i_1,i_2,\ldots,i_j,i_{j+1},i_1\}$

c) Repeat steps (a) and (b) until $T_n$ is obtained.

As it has been stated in Rosenkrantz, Stearns and Lewis [44] the fastest program devised for this method runs in a time proportional to $n^2 \log n$.

2.3.5.1.1.3 Farthest Insertion Method

Contrary to the nearest insertion method, the farthest insertion method inserts nearby nodes late in the approximation. Intuitively, the reason for such an approach is simple in the sense that the smallest distant arcs used late in the approximation have more chance of not being deleted by the later insertions. Eventually, it has been observed that this method performs well in comparison with the previously mentioned insertion methods. The algorithm is the same as the nearest insertion method except that the farthest insertion method is associated with maximization whereas the nearest insertion method is associated with minimization. The algorithm is as follows:

Given a subtour $T_j = \{i_1,\ldots,i_j,i_1\}$ with cardinality j construct another subtour $T_{j+1}$ as mentioned in the following steps.

a) Find a node $i_k \in T_j$ such that

$$C_{i_k,r} = \max_{q\in N-\{T_j\}} \{\min_{s=1,\ldots,j} \{C_{i_s,r}\}\}$$

b) Delete arc $(i_k,i_{k+1})$ in $T_j$ and add arcs $(i_k,r)$ and $r,i_{k+1})$ to obtain the subtour $T_{j+1} = \{i_1,i_2,\ldots,i_j,i_{j+1},i_1\}$.

c) Repeat steps (a) and (b) until $T_n$ is obtained.

Needless to say, the running time associated with this method is proportional to $n^2$ as it is in the nearest insertion method.

2.3.5.1.1.4 Geometric Approaches

All of the presented insertion methods of solving the TSP use the cost matrix directly to find an optimal or a near optimal solution to the problem. We will now show that given a travelling salesman graph, if the nodes can be located as points in a two dimensional space, then an optimal or at least satisfactory tour of all nodes can be obtained without reference to the cost matrix. However, the general approach remains the same. That is, the algorithm starts with a collection of nodes which comprises a partial tour and then decides which of the remaining nodes are to be inserted between which consecutive pair of nodes on this subtour and in what order. Knowing that the order of the nodes on the convex hull is the same as the order of the nodes on the optimal tour, the algorithm starts with a partial tour containing those nodes on the convex hull or with the convex hull itself.

Actually, there are two important factors which determine the efficiency and performance of the algorithms that fall into this category. First, the convex hull must be determined in order to obtain the starting partial tour. Next a criterion for choosing the next node to be inserted must be determined. The determination of the latter have caused researchers to develop different algorithms which are efficient in terms of both computational time and the satisfactory solutions obtained. The following algorithms are some of the best known algorithms found in literature.

2.3.4.1.1.4.1 The Largest Angle Method

This method due to Norback and Love [45] uses the same approach mentioned above. However, the criterion for determining the next node to be inserted between two consecutive nodes on the partial tour is to measure the angles whose vertices are the nodes to be chosen and whose sides are the arcs through consecutive nodes on the partial tour. Then, the node that corresponds to the largest of these angles is chosen to be inserted between the associated consecutive nodes on the subtour. This process is repeated until a tour containing all the nodes can be found.

It has been shown by Norback and Love that the tour generated by this method may not be optimal. For instance consider the case shown in Fig. 2.1. Starting with the convex hull {1,2,5,1} the tour obtained by the largest angle method is given as {1,2,3,5,4,1} although the optimum tour is {1,5,3,4,2,1}.

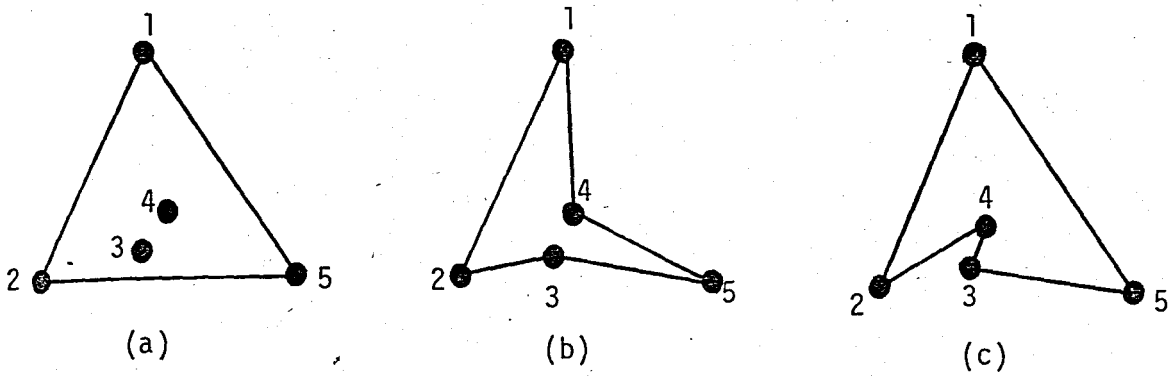



Figure 2.1 - A difficulty associated with the largest angle method

Despite of this problem the largest angle method has the special advantage of ease of application. The method has been examined to work well and fast even for large scale problems up to 2000 nodes.

2.3.5.1.1.4.2 The Most Eccentric Ellipse Method

In this method, the general approach still remains the same while the node to be inserted is being chosen by considering each consecutive pair of nodes on the convex hull as foci of an ellipse and the node to be chosen as being on the ellipse. Then, the least circular ellipse determines the node to be inserted in the subtour. An important feature of this method is that the triangle inequality is required to hold. However, considering that the nodes are points in two dimensional space, the distances between all pairs of nodes do satisfy this condition.

Nevertheless, this method may not generate an optimal tour either. The choice mechanism may fail as it does in the particular case shown in Fig. 2.2. Note that the starting hull is given as {1,2,4,3,1} and the most eccentric ellipse method inserts node 5 between nodes 2 and 4 whereas a less costly tour can be obtained

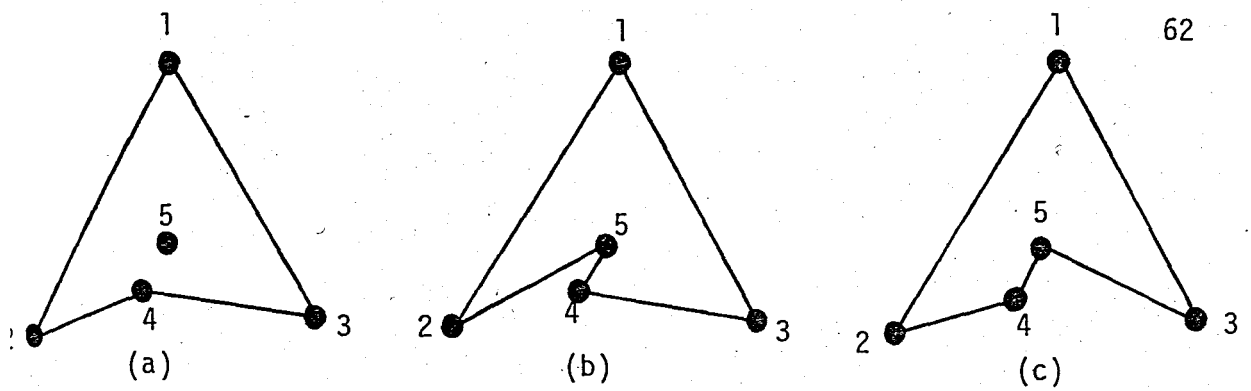



Fig. 2.2 - A difficulty associated with the most eccentric ellipse method

by inserting node 5 between nodes 4 and 3.

2.3.5.1.1.4.3 Other Convex Hull Algorithms

Analysing the geometric approaches mentioned above, we see that in particular the nodes of the travelling salesman graphs are required to be modelled as points in a two dimensional space and that the existence of the triangle inequality is a must. On the other hand, once the convex hull is known the cost matrix can be used to determine the next node to be inserted in the subtour. Again, the criteria used in determining the successive nodes to be sequenced are important and affect the efficiency and the performance of the algorithm. As it has been proposed by Or [46] using the cost matrix, three different measures can provide a means of finding out the next node to be inserted between any two consecutive nodes on the particular tour.

Let i and j be any two consecutive nodes on the convex hull, and let k be one of the remaining nodes to be inserted. Then, the measures can be defined as follows:

i) $DIST = \min\{C_{ik} + C_{kj} - C_{ij}\}$

ii) $RATIO = \min\{(C_{ik} - C_{kj})/C_{ij}\}$

iii) $MULT = DIST \times RATIO$

Experimental results showed that the best solutions were obtained when the third criterion was applied in the algorithm. Although the reason for such a result has not been determined, the fact that the third criterion is a good way of breaking ties that may occur when the first two measures are applied has been accepted to affect the solution. That is, ties that may occur in the first two measures are probable not to occur in the third one.

After all, neither of the above criteria guarantees that the optimal solution will be found. But, observations reveal that the algorithms work efficiently and obtain satisfactory results. An advantage of applying these measures is that the cost matrix need not satisfy the triangle inequality. However, since the structure of the algorithm is dependent on the topographic structure of the problem, it may generate tours which are far from being optimal for cost matrices containing arbitrary numbers. Eventually, the algorithm can also be applied to situations where the nodes are modelled as points in two dimensional space. Once the coordinates of the nodes are known, it is possible to obtain the associated costs by using the distance formula.

2.3.5.1.2 Nearest Merging Method

This method is different from the previous methods in the sense that it first constructs a set of subtours covering all the nodes and then merges two subtours at each iteration until a tour including all the nodes is constructed. In summary, the algorithm is as follows:

1. Let $S_1$ be a set of n tours each containing a single node
   Set i = 1.

2. Find an arc (q,r) such that

   $$C_{qr} = \min\{C_{kj} \text{ for k and j in different subtours in } S_i\}$$

3. Obtain $S_{i+1}$ from $S_i$ by merging two subtours containing q and r. Let those subtours be $T_1$ and $T_2$ respectively. Then, the merging process is performed as follows:

   a) If $T_1$ consists of a single node, q, then insert q into $T_2$, else if $T_2$ consists of a single node, r, then insert r into $T_1$.

   b) If $T_1$ and $T_2$ each contain at least two nodes then let s and t be nodes such that s is in $T_1$ and t is in $T_2$ and $C_{qr} + C_{st} - C_{qs} - C_{rt}$ is minimized. Delete arcs (qs) and (r,t) and add arcs (q,r) and (s,t) so that $T_1$ and $T_2$ are merged. Set i = i+1.

4. Repeat steps (2) and (3) until $S_n$ contains one tour including all nodes.

This algorithm is also bounded with a ratio similar to other insertion algorithms. That is, the ratio of the approximated tour cost to the optimal tour cost is less than 2.

2.3.5.1.3 The Nearest Neighbour Algorithm

This algorithm starts with an arbitrary node and builds up a path sequentially. Finally, the path is completed to a circuit by

adding an arc joining its end points. The algorithm uses the following steps:

1. Start with an arbitrary node.
2. Find the node not yet on the path and which is the closest to the node last added. Add the arc connecting these two nodes to the path (Ties are broken arbitrarily).
3. When all nodes have been added to the path add the arc connecting the two end nodes so that the path is completed to a circuit.

As it has been stated by Rosenkrantz, Stearns and Lewis [45] this algorithm can be programmed to operate in a time proportional to $n^2$. A possible improvement of the method is to repeat the algorithm for each possible starting node. As a result, the running time will be proportional to $n^3$. Furthermore the ratio of the approximate tour cost to the optimal tour cost is less than $((1/2)\ln(n) + (1/2))$. Note that the bounds found for all these algorithms are for their worst case behaviour. However, experiments suggest that the performance of the methods are far from being tied to their worst case behaviour.

### 2.3.5.2 Successive Improvement Techniques

Another approach to finding a satisfactory solution to the TSP is to start with a travelling salesman tour and perturb it to see if a better tour can be obtained. If a better tour is obtained, then the initial tour is discarded and the new tour is further manipulated. The procedure is repeated until no more improvement can be made and hence, the tour at hand is the best achievable solution.

The first method that we will analyse was first exploited by Croes [47]. The algorithm makes use of the important result that if the cost matrix of a travelling salesman graph represents Euclidean distances than the optimal tour does not intersect itself. Once an arbitrary tour is selected initially, the algorithm tries to produce an intersectionless tour by replacing two arcs in the tour by two other arcs that are not in the tour.

The method of local optimization was further carried by Reiter and Sherman [48]. Their algorithm starts with an arbitrary tour and tries to find the best location of each node separately. In other words, once a node is removed from the tour, the algorithm tries to find its best location in the remaining sequence. The procedure is continued until no improvement in the tour is possible. Then, the algorithm tries to find the best location of an arc joining two nodes in the sequence. For example, the location of the arc $(i_1,i_2)$ is tried to be found in the remaining sequence $\{i_3,i_4,\ldots,i_n\}$. This procedure is also continued until no improvement is possible. Finally, the algorithm checks chains of three nodes in alternative locations.

A similar approach was introduced by Lin [49] who generalized the local optimization methods. Lin defines a tour to be r-optimal if the deletion of r arcs and their replacement by other r arcs produces no better tour. Starting with an arbitrary tour, if r arcs are removed from the tour then r disconnected paths are produced. These paths can be connected in one are more different ways to produce another tour with a better total cost. As far as r-optimality is concerned, the method exploited by Croes would be called 2-optimal since it tries to obtain an improvement by interchanging any two arcs by another set of two arcs.

It is shown by Rosenkrantz, Stearns and Lewis [44] that for $n \geq 8$ there exists a graph having a tour which is r-optimal for all $r \leq n/4$ and for which the cost of that tour satisfies

$$\frac{TC(r\text{-opt})}{TC_{TSP}} = 2(1 - \frac{1}{n})$$

where TC(r-opt) is the cost of an r-optimal tour and $TC_{TSP}$ is the cost of the optimal solution to the TSP.

An important feature of this method is that the number of calculations required to obtain an r-optimal tour is polynomial in n while it is exponential in r. Therefore, only small values of r can be used in the algorithm. Note that the TSP is n-optimal and the number of operations required to obtain the n-optimal tour is (n-1)! for asymmetric problems and (n-1)!/2 for symmetric problems. In fact, this is the quantity required for the complete enumeration of all the possible tours. The method was further improved by Lin and Kernighan [50] in a more powerful way.

Another successive improvement technique for finding an approximat tour is accomplished by first starting with any tour and then trying to switch the position of the nodes. Let $\{i_1, i_2, \ldots, i_n\}$ denote the order of the nodes in the initial tour. Then, the algorithm tries to find a shorter tour by switching each possible pair of nodes in the tour. Switching nodes $i_j$ and $i_k$ means replacing arcs $(i_{j-1}, i_j)$, $(i_j, i_{j+1})$, $(i_{k-1}, i_k)$, $(i_k, i_{k+1})$ by arcs $(i_{j-1}, i_k)$, $(i_k, i_{j+1})$, $(i_{k-1}, i_j)$, $(i_j, i_{k+1})$. The switching procedure continues until no improvement is possible [51].

As in all improvement techniques the final tour depends on the initial tour. Moreover, the cost of the initial tour should not be considered as a good indicator of the cost of the final tour. In general, one cannot be certain about the optimality of the final tour produced by these methods. But they are known to perform well in most of the cases.

### 2.3.5.3 Techniques Using Minimal Spanning Trees

Most of the methods which are proven to have constant bounds use comparisons with minimal spanning trees in their proofs. Then, a question that may come into mind is "why shouldn't minimal spanning trees be used in finding approximate solutions to the TSP?" Eventually, there are widely known methods which determine approximate solutions by first finding the minimal spanning trees. We will analyse some of these methods and outline their algorithms to give an insight to the use of the minimal spanning trees in finding approximate solutions rather than using them as lower bounds to the TSP as has been explained previously.

The first method we will analyse is the penalty method introduced by Christofides [24]. The spirit of this algorithm is to transform the cost matrix in such a way that the minimal spanning tree of the transformed matrix is forced to form a Hamiltonian path. The algorithm proceeds as follows:

1. Find the minimal spanning tree of G = (N,E) using the cost matrix C.
2. If the minimal spanning tree is a Hamiltonian path then the problem is solved. If not, then calculate a penalty, $\bar{p}_i$, for each node i and transform the cost matrix such that

$$C_{ij} = C_{ij} + \bar{p}_i + \bar{p}_j \qquad \forall i,j \in N$$

3. Repeat steps (1) and (2) until a Hamiltonian (shortest) path is found.

4. Add the arc joining the two ends of the Hamiltonian path to produce a travelling salesman tour.

There are many strategies for computing the associated penalties at each step. Held and Karp [14] who developed a similar algorithm gave two methods of finding the penalties which minimize the difference between the cost of the shortest Hamiltonian path and the cost of the minimal spanning tree under the modified cost matrices. A pitfall of this algorithm, however, is that it is not necessarily convergent. But it can be considered as a valuable method since it converges in the great majority of the cases. Moreover, it may be used as a valid lower bound in cases when the algorithm does not converge.

Another widely known but unpublished method using minimal spanning trees is as follows [44]:

1. Find the minimal spanning tree of the graph.

2. Double the arcs of the minimal spanning tree so that an Eulerian circuit containing each node at least once is obtained.

3. Construct a travelling salesman tour by traversing the arcs of the Eulerian circuit (i.e. a circuit traversing each of the arcs at least once). If a node already included in the travelling salesman tour appears in the sequence of the Eulerian tour, skip that node and continue traversing until all the nodes are included in the travelling salesman tour.

This method also has a ratio of the obtained tour cost to the optimal tour cost which is less than 2.

Christofides [52] developed a similar algorithm which give a better bound for the worst case behaviour. A worst case analysis of his heuristic showed that the bounding ratio is strictly less than (3/2). This brought a 50% reduction over the previously best known ratios for other polynomially bounded algorithms. The algorithm can be stated as follows:

1. Find the minimal spanning tree of the graph $G = (N,E)$.
2. Relative to the minimal spanning tree, let $N^1$ be the set of nodes having odd degree. Solve the 1-matching problem for the graph $G^1 = (N^1,E^1)$.
3. Let only those arcs in the minimal spanning tree and those arcs in the matching and the set of nodes N comprise the graph $G^2 = (N,E^2)$. This graph has all nodes of even degree and consequently possesses an Eulerian circuit.
4. Transform the Eulerian circuit into a travelling salesman tour by removing extra occurrences of each node.

Several good algorithms exist for finding the minimal spanning tree of a graph. Usually, these algorithms have a computational time which is of order $O(n^2)$. However, the best known algorithms for finding the minimum matching have a computational growth rate $O(n^3)$. Therefore, the overall computational time is proportional to $n^3$. Note that, the last step of converting the Eulerian circuit to a Hamiltonian circuit

can be done in linear time. After all, the best known bound has been improved by 50% in the expense of increasing the computational effort with regard to methods which have worse bounds but have computational effort which is proportional to $n^2$.

A final remark that should be made for the methods that fall into this category is that techniques using minimal spanning trees are applicable only to symmetric TSPs. This is a consequence of the fact that minimal spanning trees can only be computed for undirected graphs.

# III. FOUR HEURISTIC ALGORITHMS FOR SOLVING THE TRAVELLING SALESMAN PROBLEM

In this chapter we introduce four heuristic algorithms using four distinct approaches for solving the TSP. Similar to all of the heuristic methods that have been put forward recently the algorithms are designed so that they do not suffer from inefficiency. Actually, the algorithms are easily programmable on a computer and produce tours which are close to the optimal solutions.

The first algorithm uses the necessary conditions for the existence of a Hamiltonian circuit as a tool for constructing a subgraph of the original graph in which the optimum or a near optimum solution to the TSP is contained. The arc set of the subgraph is extended by including arcs corresponding to the zero cost elements in the updated cost matrix as a result of reducing it iteratively. The reduction is made in such a way that the necessary conditions for the existence of a Hamiltonian circuit tend to hold as the subgraph is developed. In case, the subgraph does not contain any Hamiltonian circuit, the algorithm applies Little's branch and bound algorithm to the resultant matrix partially so that a feasible tour is obtained.

The fact that the travelling salesman tours are extreme points of the assignment polytope constitutes the main idea used in the second algorithm. The TSP is solved by the aid of the embedded assignment problems. The subtours produced by the assignment solutions are broken is such a way as to make the algorithm work as fast as possible.

The third algorithm uses a dynamic programming type approach which is very similar to Ford's [53] shortest path algorithm. First, all the elements of the cost matrix are subtracted from a large number so that the triangle inequality is satisfied. Then, given a specified root node, the algorithm tries to find all the longest Hamiltonian paths in which all the nodes appear once and only once. The longest paths are then completed to Hamiltonian circuits and the one with the least cost (i.e. calculated by using the original cost matrix) is selected as the best achievable solution.

The last algorithm is a geometric approach which uses the well known tour building technique. Similar to other relevant algorithms, the method works well for problems defined in the Euclidean space. Given the convex hull, the algorithm calculates the heights of the triangles whose bases are the arcs through consecutive node pairs in the convex hull and whose third vertices are the interior nodes that are not in the convex hull. As a result, the heights are considered as a measure of inserting the interior nodes and therefore building the final tour.

Repeated runs on randomly generated graphs resulted in finding solutions which are near optimal. The heuristics showed different growth rates in the computation effort all of which are as substantially well as other existing algorithms which use the same approaches.

## 3.1 ALGORITHM I

Given an undirected (directed) graph G = (N,E), the algorithm constructs a subgraph G' = (N,E') of G which is (strongly) connected. Then, the arc set of G' is extended in such a way that each subgraph $G'_k$ constructed by removing a node from G' is unilaterally connected.

The algorithm starts with reducing the associated cost matrix C. That is, the minimum element of each row is subtracted from all the elements of that row and the minimum element of each column is subtracted from all the elements of that column. As a result, all the arcs (i,j) with $C_{ij} = 0$ comprise the arc set E' of G'. Then the (strong) connectedness of G' is checked. If the graph is not (strongly) connected the cost matrix is further reduced in a sequential manner so that other arcs with $C_{ij} = 0$ can be included in E' and therefore (strong) connectedness can be achieved. Note that (strong) connectedness is necessary for the existence of a Hamiltonian circuit, i.e. a solution to the TSP, but is not sufficient.

Consider that there exists a Hamiltonian circuit in G'. If a node k is removed from G', the resulting subgraph $G'_k$ contains a Hamiltonian path through the remaining (n-1) nodes and is, therefore, a unilaterally connected subgraph. In other words, for any two nodes i and j of $G'_k$, there exists a path either from i to j or from j to i. Hence, unilateral connectedness in $G'_k$ is also a necessary condition for the existence of a Hamiltonian circuit. As a consequence of this fact, for every excluded node $k \in N$, the algorithm checks if the resultant subgraph $G'_k$ is unilaterally connected. If not, the cost matrix is reduced iteratively until the condition holds.

As a matter of fact, since both of the conditions checked are not sufficient conditions the resultant subgraph G' may still not contain any Hamiltonian circuit. Experiments show that this is true especially when n, i.e., the number of nodes gets larger. In that case, a procedure which incorporates the first part of Little's branch and bound algorithm is applied to the resultant matrix. However, the branch and bound procedure is never used completely. The procedure stops as soon as a feasible solution is found. The solution is assumed to be the best one that can be obtained. More formally, the algorithm can be expressed as follows:

1. Reduce the rows and columns of the cost matrix, C, such that each row and column has at least one zero. Let all the arcs $(i,j)$ with $C_{ij} = 0$ comprise the arc set of $G' = (N,E')$.
2. Check for (strong) connectedness. If G' is not (strongly) connected then apply the following steps to achieve (strong) connectedness:
   a) Choose a node $k \in N$
   b) Find the node set R(k) which can be reached from node k by means of path in G'. Let node k be included in R(k).
   c) If $R(k) \neq N$, then find

$$\pi_k = \min_{\substack{i \in R(k) \\ j \in N-R(k)}} \{C_{ij}\}$$

   and update the cost matrix by

$$C_{ij} = C_{ij} - \pi_k \qquad \forall i \in R(k), \quad j \in N-R(k)$$

include new arcs having $C_{ij} = 0$ into E' return to step (b). Otherwise, continue.

d) If R(k) = N, then choose another node $k \in N$ among the ones which have not been checked yet. If all the nodes have been checked, then continue with step (3). Otherwise return to step (b).

3. For each node $k \in N$, perform the following steps:

a) Let $G'_k = (N-\{k\}, E'_k)$ be the subgraph obtained by removing node k and the arcs entering and emanating from node k in G'.

b) Find the set of nodes R(i) which can be reached from node i via arcs in $G'_k$ for every $i \in N-\{k\}$. Check if $G'_k$ is unilaterally connected by going through the following:

i) Choose a node $q \in N-\{k\}$.

ii) If $R(q) \neq N-\{k\}$, then for all nodes $r \in N-\{k\}-R(q)$ check if $q \in R(r)$. In other words, check if there exists a path from r to q for all q which are not reachible from r. If such a node q is found, let

$$\mu_{qr} = \min\{ \min_{\substack{i\in R(q) \\ j\in N-\{k\}-R(q)}} \{C_{ij}\}, \min_{\substack{i\in R(r) \\ j\in N-\{k\}-R(r)}} \{C_{ij}\}\}$$

Update the cost matrix by

$$C_{ij} = C_{ij} - \mu_{qr} \qquad \begin{array}{l} \forall i \in R(q) \ , \ j \in N-\{k\}-R(q) \\ \forall i \in R(r) \ , \ j \in N-\{k\}-R(r) \end{array}$$

Include new arcs (i,j) with $C_{ij} = 0$ in E' and $E'_k$.

Define the reachible sets R(i) again and repeat this step. If either R(q) = N-{k} or no node r such that $r \notin R(q)$ and $q \notin R(r)$ can be found, then choose another node q among the ones which have not been tried yet. Repeat this step until all nodes are considered.

4. As a result of performing steps (1) through (3), the necessary conditions for the existence of a Hamiltonian circuit in G' are satisfied. Therefore, check if G' contains a Hamiltonian circuit. If so, evaluate all the Hamiltonian circuits in G' and select the one with the least cost as the best achievable solution to the TSP. Otherwise continue.

5. Apply Little's branch and bound algorithm partially to the resultant matrix until a feasible solution can be obtained.

The steps of Little's branch and bound algorithm are not repeated here since the whole algorithm is given in Chapter 2. However, it should be noded that, no branching procedure is performed if the cost matrix corresponds to a complete graph. Moreover, it is highly probable that the branching procedure would not be performed for graphs that are not complete either.

*Example 3.1*

Consider a complete directed graph whose cost matrix is given in Table 3.1a and suppose that we want to solve the TSP by the use of the algorithm described above. Applying the first step of the algorithm, the reduced matrix and the corresponding subgraph G' are shown

in Table 3.1b and Fig. 3.1a respectively. As it can be seen G' is disconnected and therefore we proceed by applying step (2).

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | ∞ | 76 | 43 | 38 | 51 | 42 | 19 | 80 |
| 2 | 42 | ∞ | 49 | 26 | 78 | 52 | 39 | 87 |
| 3 | 49 | 28 | ∞ | 36 | 53 | 44 | 68 | 61 |
| 4 | 72 | 31 | 29 | ∞ | 42 | 49 | 50 | 38 |
| 5 | 30 | 52 | 38 | 47 | ∞ | 64 | 75 | 82 |
| 6 | 66 | 51 | 83 | 51 | 22 | ∞ | 37 | 71 |
| 7 | 77 | 62 | 93 | 54 | 69 | 38 | ∞ | 26 |
| 8 | 42 | 58 | 66 | 76 | 41 | 52 | 83 | ∞ |

(a)

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | ∞ | 57 | 24 | 19 | 32 | 12 | 0 | 61 |
| 2 | 16 | ∞ | 23 | 0 | 52 | 15 | 13 | 61 |
| 3 | 20 | 0 | ∞ | 8 | 25 | 5 | 40 | 33 |
| 4 | 43 | 2 | 0 | ∞ | 13 | 9 | 21 | 9 |
| 5 | 0 | 14 | 0 | 9 | ∞ | 23 | 45 | 52 |
| 6 | 44 | 21 | 53 | 21 | 0 | ∞ | 15 | 49 |
| 7 | 51 | 28 | 59 | 20 | 43 | 1 | ∞ | 0 |
| 8 | 1 | 8 | 17 | 27 | 0 | 0 | 42 | ∞ |

(b)

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | ∞ | 49 | 16 | 11 | 32 | 12 | 0 | 61 |
| 2 | 11 | ∞ | 23 | 0 | 47 | 10 | 8 | 56 |
| 3 | 15 | 0 | ∞ | 8 | 20 | 0 | 35 | 28 |
| 4 | 38 | 2 | 0 | ∞ | 8 | 4 | 16 | 4 |
| 5 | 0 | 14 | 0 | 9 | ∞ | 23 | 45 | 52 |
| 6 | 44 | 21 | 53 | 21 | 0 | ∞ | 15 | 49 |
| 7 | 51 | 28 | 59 | 20 | 43 | 1 | ∞ | 0 |
| 8 | 1 | 8 | 17 | 27 | 0 | 0 | 42 | ∞ |

(c)

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | ∞ | 49 | 16 | 11 | 32 | 12 | 0 | 61 |
| 2 | 11 | ∞ | 23 | 0 | 47 | 10 | 8 | 56 |
| 3 | 15 | 0 | ∞ | 8 | 20 | 0 | 35 | 28 |
| 4 | 38 | 2 | 0 | ∞ | 8 | 4 | 16 | 4 |
| 5 | 0 | 14 | 0 | 9 | ∞ | 23 | 45 | 52 |
| 6 | 44 | 21 | 53 | 21 | 0 | ∞ | 15 | 49 |
| 7 | 51 | 28 | 59 | 20 | 43 | 1 | ∞ | 0 |
| 8 | 1 | 8 | 17 | 27 | 0 | 0 | 42 | ∞ |

(d)

Table 3.1 - Reduced matrices obtained during the application of steps (1) through (4) of algorithm I

Table 3.1 continued.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | ∞ | 45 | 16 | 7 | 32 | 12 | 0 | 61 |
| 2 | 7 | ∞ | 23 | 0 | 43 | 6 | 4 | 52 |
| 3 | 15 | 0 | ∞ | 8 | 20 | 0 | 35 | 28 |
| 4 | 34 | 2 | 0 | ∞ | 4 | 0 | 12 | 0 |
| 5 | 0 | 10 | 0 | 5 | ∞ | 23 | 45 | 52 |
| 6 | 44 | 17 | 53 | 17 | 0 | ∞ | 15 | 49 |
| 7 | 51 | 24 | 59 | 16 | 43 | 1 | ∞ | 0 |
| 8 | 1 | 4 | 17 | 23 | 0 | 0 | 42 | ∞ |

(e)

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | ∞ | 44 | 15 | 6 | 32 | 12 | 0 | 61 |
| 2 | 6 | ∞ | 23 | 0 | 43 | 6 | 3 | 52 |
| 3 | 14 | 0 | ∞ | 8 | 20 | 0 | 34 | 28 |
| 4 | 33 | 2 | 0 | ∞ | 4 | 0 | 11 | 0 |
| 5 | 0 | 10 | 0 | 5 | ∞ | 23 | 45 | 52 |
| 6 | 43 | 16 | 52 | 16 | 0 | ∞ | 14 | 49 |
| 7 | 51 | 23 | 58 | 15 | 43 | 1 | ∞ | 0 |
| 8 | 0 | 3 | 16 | 22 | 0 | 0 | 41 | ∞ |

(f)

Choose node 1. $R(1) = \{1,5,6,7,8\} \neq N$

$$\pi_1 = \min_{\substack{i \in R(1) \\ j \in N-R(1)}} \{C_{ij}\} = C_{53} = 8$$

The updated cost matrix is given in Table 3.1c. Including arc (5,3) in G', (Fig. 3.1b) we see that R(1) = N.

Choose node 2. $R(2) = \{2,3,4\} \neq N$

$$\pi_2 = \min_{\substack{i \in R(2) \\ j \in N-R(2)}} \{C_{ij}\} = C_{36} = 5$$

Once arc (3,6) is included in G', we obtain R(2) = N. The resultant matrix is given in Table 3.1d and G' is shown in Fig. 3.1c.

Choose node 3. R(3) = N

Choose node 4. R(4) = N

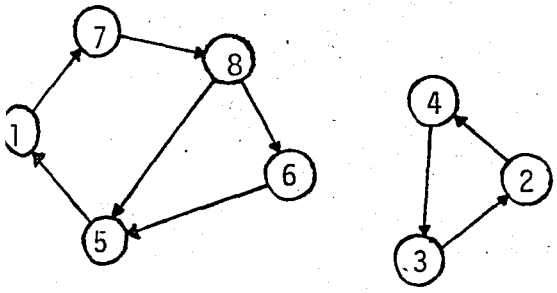
Choose node 5. R(5) = N

Choose node 6. R(6) = N

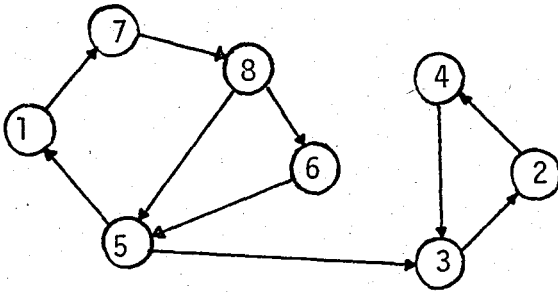
Choose node 7. R(7) = N

Choose node 8. R(8) = N

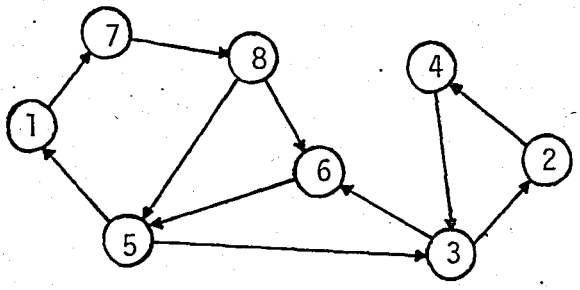
Since all the nodes have been tried and R(i) = N, $\forall i \in N$, (strong)




(a)




(b)




(c)

Figure 3.1 - Stages in constructing the subgraph G' for Example 3.1.

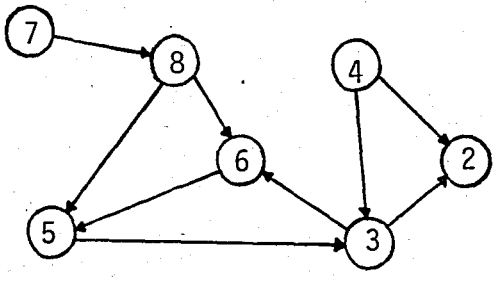



Figure 3.2a - Subgraph $G'_1$

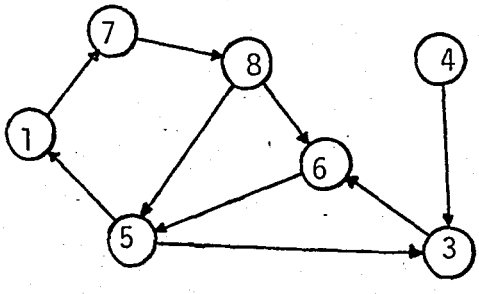



Figure 3.2b - Subgraph $G'_2$

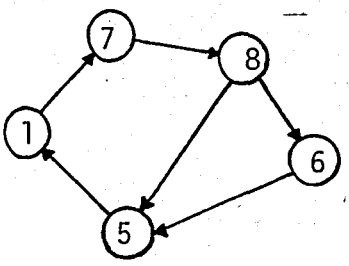
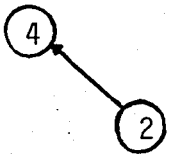



Figure 3.2c - Subgraph $G_3'$



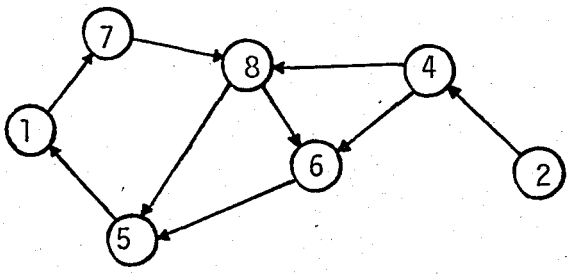




Figure 3.2d - Subgraph $G_3'$

connectedness is achieved. We proceed with step 3 to see if G' is unilaterally connected after removing any node k

Remove node 1. $G_1'$ is unilaterally connected (Fig. 3.2a).

Remove node 2. $G_2'$ is unilaterally connected (Fig. 3.2b).

Remove node 3. $G_3'$ is not unilaterally connected (Fig. 3.2c).

There is not a path either from node 1 to 2 or from 2 to 1.

$R(1) = \{1,5,6,7,8\}$ $\quad$ $R(2) = \{2,4\}$

$N-\{3\}-R(1) = \{2,4\}$ $\quad$ $N-\{3\}-R(2) = \{1,5,6,7,8\}$

$$\mu_{12} = \min\{ \min_{\substack{i\in R(1) \\ j\in N-\{3\}-R(1)}} \{C_{ij}\}, \min_{\substack{i\in R(2) \\ j\in N-\{3\}-R(2)}} \{C_{ij}\}\} = C_{46} = C_{48} = 4$$

The resultant cost matrix is given in Table 3.1c and and the updated $G_3'$ is shown in Fig. 3.2d. Note that $G_3'$ becomes unitalerally connected.

Remove node 4. $G_4'$ is unilaterally connected (Fig. 3.2e).

Remove node 5. $G_5'$ is not unilaterally connected (Fig. 3.2f).

There is not a path either from node 1 to 2 or from 2 to 1.

$R(1) = \{1,6,7,8\}$ $\quad$ $R(2) = \{2,3,4,6,8\}$

$N-\{5\}-R(1) = \{2,3,4\}$ $\quad$ $N-\{5\}-R(2) = \{1,7\}$

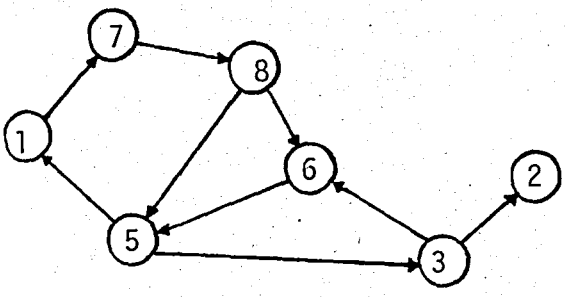



Figure 3.2e - Subgraph $G'_4$

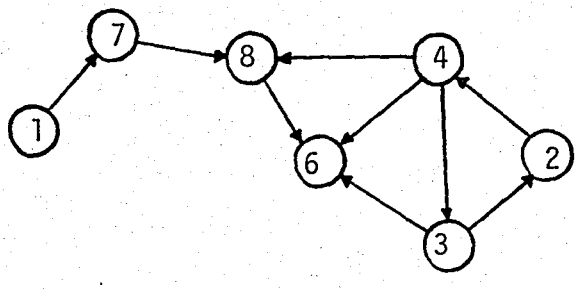



Figure 3.2f - Subgraph $G'_5$

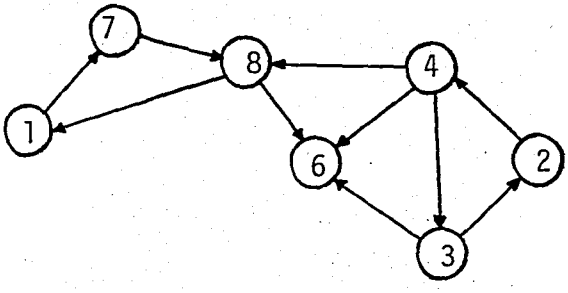



Figure 3.2g - Subgraph $G'_5$

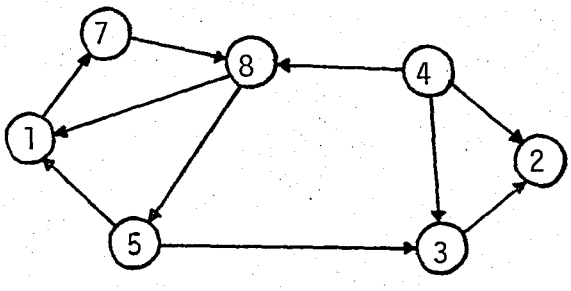



Figure 3.2h - Subgraph $G'_6$

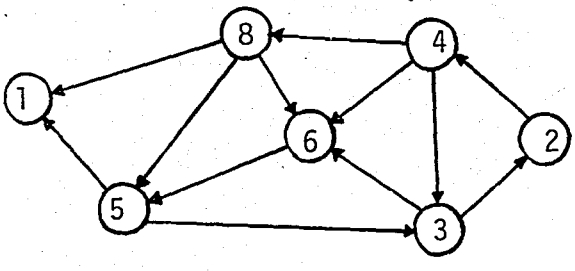



Figure 3.2i - Subgraph $G'_7$

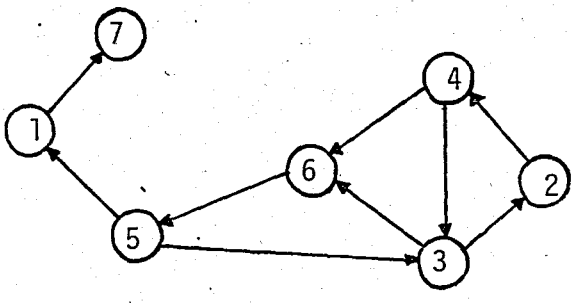



Figure 3.2j - Subgraph $G'_8$

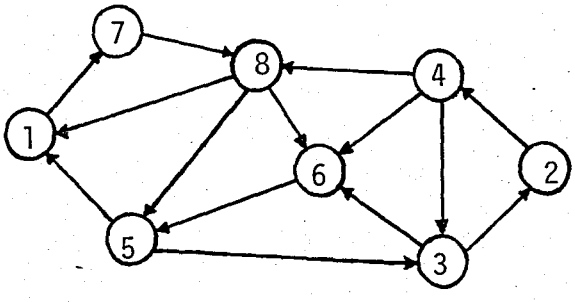



Figure 3.3 - Resultant subgraph G'

$$\mu_{12} = \min\{ \min_{\substack{i\in R(1) \\ j\in N-\{5\}-R(1)}} \{C_{ij}\} , \min_{\substack{i\in R(2) \\ j\in N-\{5\}-R(2)}} \{C_{ij}\}\} = C_{81} = 1$$

The cost matrix is updated as shown in Table 3.1f. As a result, $G_5'$ becomes unilaterally connected (Fig. 3.2g).

Remove node 6. $G_6'$ is unilaterally connected (Fig. 3.2h).

Remove node 7. $G_7'$ is unilaterally connected (Fig. 3.2i).

Remove node 8. $G_8'$ is unilaterally connected (Fig. 3.2j).

The resultant subgraph G' is shown in Fig. 3.3. Note that G' does not possess any Hamiltonian circuit. Consequently we proceed with step (5). The steps of Little's branch and bound algorithm can be followed in Table 3.2.

Starting with the resultant matrix we calculate the associated penalties $\bar{p}_{ij}$ which correspond to entries with $C_{ij} = 0$. The maximum of the penalties (Table 3.2a) is $\bar{p}_{65} = 14$. We delete row 6 and column 5 and insert infinity to $C_{56}$. The new matrix and the associated penalties are given in Table 3.2b. At this stage, $\bar{p}_{17} = 9$ is the maximum penalty. Therefore, we delete row 1 and column 5, insert infinity into $C_{71}$ and obtain the matrix in Table 3.2c. Calculating the penalties in the new matrix we choose $\bar{p}_{24} = 11$ as being the maximum one. We delete row 2 and column 4 and insert infinity to $C_{42}$. The induced matrix and the corresponding penalties are given in Table 3.2d. As a result, we choose the penalty $\bar{p}_{32} = 3$ and delete row 3 and column 2. Note that we have to insert infinity into $C_{43}$ in order to prevent the subloop (3-2-4-3). Once we obtain the new matrix and calculate the associated penalties (Table 3.2e) the maximum penalty corresponds to

Table 3.2a

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | ∞ | 44 | 15 | 6 | 32 | 12 | 0 | 61 |
| 2 | 6 | ∞ | 23 | 0 | 43 | 6 | 3 | 52 |
| 3 | 14 | 0 | ∞ | 8 | 20 | 0 | 34 | 28 |
| 4 | 33 | 2 | 0 | ∞ | 4 | 0 | 11 | 0 |
| 5 | 0 | 10 | 0 | 5 | ∞ | 23 | 45 | 52 |
| 6 | 43 | 16 | 52 | 16 | 0 | ∞ | 14 | 49 |
| 7 | 51 | 23 | 58 | 15 | 43 | 1 | ∞ | 0 |
| 8 | 0 | 3 | 16 | 22 | 0 | 0 | 41 | ∞ |

$\overline{p}_{17} = 9$ $\overline{p}_{46} = 0$ $\overline{p}_{78} = 1$
$\overline{p}_{24} = 8$ $\overline{p}_{48} = 0$ $\overline{p}_{81} = 0$
$\overline{p}_{32} = 2$ $\overline{p}_{51} = 0$ $\overline{p}_{85} = 0$
$\overline{p}_{36} = 0$ $\overline{p}_{53} = 0$ $\overline{p}_{86} = 0$
$\overline{p}_{43} = 0$ $\overline{p}_{65} = 14$

Table 3.2b

| | 1 | 2 | 3 | 4 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 1 | ∞ | 44 | 15 | 6 | 12 | 0 | 61 |
| 2 | 6 | ∞ | 23 | 0 | 6 | 3 | 52 |
| 3 | 14 | 0 | ∞ | 8 | 0 | 34 | 28 |
| 4 | 33 | 2 | 0 | ∞ | 0 | 11 | 0 |
| 5 | 0 | 10 | 0 | 5 | ∞ | 45 | 52 |
| 7 | 51 | 23 | 58 | 15 | 1 | ∞ | 0 |
| 8 | 0 | 3 | 16 | 22 | 0 | 41 | ∞ |

$\overline{p}_{17} = 9$ $\overline{p}_{43} = 0$ $\overline{p}_{53} = 0$
$\overline{p}_{24} = 8$ $\overline{p}_{46} = 0$ $\overline{p}_{78} = 1$
$\overline{p}_{32} = 2$ $\overline{p}_{48} = 0$ $\overline{p}_{81} = 0$
$\overline{p}_{36} = 0$ $\overline{p}_{51} = 0$ $\overline{p}_{86} = 0$

Table 3.2c

| | 1 | 2 | 3 | 4 | 6 | 8 |
|---|---|---|---|---|---|---|
| 2 | 6 | ∞ | 23 | 0 | 6 | 52 |
| 3 | 14 | 0 | ∞ | 8 | 0 | 28 |
| 4 | 33 | 2 | 0 | ∞ | 0 | 0 |
| 5 | 0 | 10 | 0 | 5 | ∞ | 52 |
| 7 | ∞ | 23 | 58 | 15 | 1 | 0 |
| 8 | 0 | 3 | 16 | 22 | 0 | ∞ |

$\overline{p}_{24} = 11$ $\overline{p}_{46} = 0$ $\overline{p}_{78} = 1$
$\overline{p}_{32} = 2$ $\overline{p}_{48} = 0$ $\overline{p}_{81} = 0$
$\overline{p}_{36} = 0$ $\overline{p}_{51} = 0$ $\overline{p}_{36} = 0$
$\overline{p}_{43} = 0$ $\overline{p}_{53} = 0$

Table 3.2d

| | 1 | 2 | 3 | 6 | 8 |
|---|---|---|---|---|---|
| 3 | 14 | 0 | ∞ | 0 | 28 |
| 4 | 33 | ∞ | 0 | 0 | 0 |
| 5 | 0 | 10 | 0 | ∞ | 52 |
| 7 | ∞ | 23 | 58 | 1 | 0 |
| 8 | 0 | 3 | 16 | 0 | ∞ |

$\overline{p}_{32} = 3$ $\overline{p}_{48} = 0$ $\overline{p}_{81} = 0$
$\overline{p}_{36} = 0$ $\overline{p}_{51} = 0$ $\overline{p}_{86} = 0$
$\overline{p}_{43} = 0$ $\overline{p}_{53} = 0$
$\overline{p}_{46} = 0$ $\overline{p}_{78} = 1$

Table 3.2e

| | 1 | 3 | 6 | 8 |
|---|---|---|---|---|
| 4 | 33 | ∞ | 0 | 0 |
| 5 | 0 | 0 | ∞ | 52 |
| 7 | ∞ | 58 | 1 | 0 |
| 8 | 0 | 16 | 0 | ∞ |

$\overline{p}_{46} = 0$ $\overline{p}_{53} = 16$ $\overline{p}_{86} = 0$
$\overline{p}_{48} = 0$ $\overline{p}_{78} = 1$
$\overline{p}_{51} = 0$ $\overline{p}_{81} = 0$

Table 3.2f

| | 1 | 6 | 8 |
|---|---|---|---|
| 4 | 33 | ∞ | 0 |
| 7 | ∞ | 1 | 0 |
| 8 | 0 | 0 | ∞ |

$\overline{p}_{48} = 33$ $\overline{p}_{81} = 33$
$\overline{p}_{78} = 1$ $\overline{p}_{86} = 1$

Table 3.2g

| | 1 | 6 |
|---|---|---|
| 7 | ∞ | 1 |
| 8 | 0 | ∞ |

| | 1 | 6 |
|---|---|---|
| 7 | ∞ | 0 |
| 8 | 0 | ∞ |

the zero entry in $C_{53}$ with $\overline{p}_{53}$ = 16. We delete row 5 and column 3 and insert infinity into $C_{46}$ so that the subloop (6-5-3-2-4-6) is prohibited (Table 3.2f). The next maximum penalty is $\overline{p}_{48}$ = 33. After deleting row 4 and column 8 we insert infinity to $C_{86}$ in order not to allow the subloop (6-5-3-2-4-8-6) to appear in the final solution. Note that we have a (2x2) matrix at hand now. On the other hand we have to reduce the matrix in order to have at least one zero in each row and in each column (Table 3.2g). As a result, we have one choice. That is, we include arcs (7,6) and (8,1) into the solution set. Therefore, the solution obtained is (1-7-6-5-3-2-4-8-1) with a total cost of 251 which happens to be the actual optimum solution to the problem.

## 3.2 ALGORITHM II

Methods using the assignment problem (AP) have been a feasible line of attack for solving the TSP since the AP is a valid relaxation of the TSP and has a polynomially bounded solution method. It is obvious that the optimum solution to the TSP is a feasible solution to the AP since any travelling salesman tour is an assignment. Unfortunately, the reverse is not true. That is, an assignment solution is not necessarily a travelling salesman tour. However, we know that the travelling salesman tours correspond to extreme points of the assignment polytope. Therefore, ranking methods can be used to find the optimum to the TSP by solving the AP successfully.

Consider the AP with an (nxn) cost matrix C whose diagonal elements are all set to infinity. We want to rank all the assignments

in increasing order of cost until a TSP solution is obtained. This can be achieved by using a branch and bound scheme. An important operation performed on the nodes of the decision tree is that of partitioning them using the minimum cost assignment solution. Let M be the node representing the set of all solutions and

$$S_M = \{(a_1,b_1),\ldots,(a_u,b_u),(\overline{q_1,r_1}),\ldots,(\overline{q_v,r_v}),(i_1,j_1),\ldots,(i_n,j_n)\}$$

denote an optimum solution in M where the first set of u arcs are those which are required to appear in the optimum solution and the second set of v arcs are those which are not wanted to appear in the optimum solution. The last set of arcs are the optimal combination of the remaining assignments. Then, the partitioning scheme can be performed as follows:

$$M_1 = \{(a_1,b_1),\ldots,(a_u,b_u),(\overline{q_1,r_1}),\ldots,(\overline{q_v,r_v}),(\overline{i_1,j_1})\}$$

$$M_2 = \{(a_1,b_1),\ldots,(a_u,b_u),(\overline{q_1,r_1}),\ldots,(\overline{q_v,r_v}),(i_1,j_1),(\overline{i_2,j_2})\}$$

$$\vdots$$

$$M_{n-1} = \{(a_1,b_1),\ldots,(a_u,b_u),(\overline{q_1,r_1}),\ldots,(\overline{q_v,r_v}),(i_1,j_1),\ldots,(i_{n-2},j_{n-2}),(\overline{i_{n-1},j_{n-1}})\}$$

The partitioning of M using $S_M$ generates the subnodes $M_1,\ldots,M_{n-1}$ and the partition itself is

$$M = \{S_M\} \cup \left(\bigcup_{k=1}^{n-1} M_k\right)$$

Note that the subnodes $M_1,\ldots,M_{n-1}$ are all nonempty and mutually disjoint. At each stage, the algorithm maintains a list which is a set of nodes. Each node in the list is stored together with the minimum cost assignment and its objective function value. This algorithm was

first developed by Murty [54] and used to solve problems in which the minimal cost assignment satisfying additional constraints is required. One such problem is the TSP. For that case, the algorithm is initiated by finding a minimum cost assignment using the Hungarian method. Then, the set of all solutions is partitioned as mentioned above. For each subnode the corresponding minimum assignment is found. As a result, the algorithm branches to the node with the minimum cost. The procedure is continued until the assignment corresponding to the branched node is a travelling salesman tour. However, the storage requirements for the list of nodes and the associated AP solutions are considerably high. Furthermore, the number of branches in the decision tree is highly dependent on the nature of the cost matrix of the TSP. In other words, the size of the decision tree depends on the difference between the optimum solution to the TSP and the optimum solution to the AP, and therefore, on the number of extreme points in between.

The proposed algorithm provides a means of getting rid of the need of information keeping required in the algorithm presented above. Moreover, at each iteration the algorithm introduces a new cut that forces the AP solution to form a tour. Thus, exclusion of some extreme points from consideration becomes possible. The only book keeping involves the storage of the cost matrix belonging to the previous iteration. The algorithm is as follows:

1. Solve the AP. Let $Z_0$ be its objective function value.
2. If the assignment is a tour then stop. It is the optimum solution to the TSP. Otherwise continue.

3. For each arc $(i_k,j_k)$ corresponding to the assignment, calculate a minimum penalty $\overline{p}_{i_k,j_k}$ which would be incurred if that arc is not to appear in the next solution. The penalties are calculated as follows:

   a) Let $T_\ell$ be the set of nodes corresponding to the subtour in which arc $(i_k,j_k)$ is included. Calculate

   $$\overline{p}_{i_k,j_k} = \min_{r\in T_\ell} \{C_{i_k,r}\} + \min_{\substack{q\neq i_k \\ q\in N}} \{C_{q,j_k}\}$$

   b) If $\overline{p}_{i_k j_k} < 0$, then find a combination of nodes, say r and q such that

   $$\overline{p}_{i_k,j_k} = C_{i_k r} + C_{q j_k} \qquad r \notin T_\ell\ , \quad s \in N,\ s \neq i_k$$

   is a minimum positive quantity.

   c) Repeat steps (a) and (b) for every assignment $(i_k,j_k)$.

4. Starting with the minimum penalty perform the following:

   a) Insert infinities to all $C_{i_k,r}$ such that $r \in T_\ell$. Set $Z_1^* = \infty$ .

   b) Solve the AP on the updated matrix by using the Hungarian algorithm. Be careful not to perform reductions which will cause the corresponding objective function value $Z_1$ to become less than zero. Furthermore, store the reductions made in the entries into which infinities were inserted. At the end of the Hungarian algorithm add the

stored quantities to the corresponding entries. Note that, after these entries are updated the resultant values may be less than zero. The infinities are then replaced by the updated quantities so that the arcs corresponding to these entries may enter the basis later. If $Z_1 < Z_1^*$, then set $Z_1^* = Z_1$.

c) Repeat steps (a) and (b) by considering the penalties in ascending order. The procedure is continued until $Z_1^*$ is less than or equal to the next minimum penalty that will be considered. Then set $Z_0 = Z_0 + Z_1^*$ and return to step (2).

Recalling the formulation of the TSP, the unsatisfied constraints after the AP is solved are of the constraint type (2.4a). These constraints are considered while the penalties are calculated. In other words, let $T_\ell$, $\ell = 1,\ldots,q$ be the subsets of nodes corresponding to q subtours in the AP solution. Then, the unsatisfied constraints are

$$\sum_{i\in T_\ell} \sum_{j\in \overline{T}_\ell} x_{ij} \geq 1 \qquad \ell = 1,\ldots,q \qquad (3.1)$$

where $T_\ell \cup \overline{T}_\ell = N$. Eventually, for all $\overline{p}_{i_k j_k}$ such that $i_k, j_k \in T_\ell$, $\ell = 1,\ldots,q$, the quantity

$$\min_{r\notin T_\ell} \{C_{i_k,r}\}$$

determines the corresponding zero-one variable $x_{i_k,r}$ that will probably enter the basis and therefore satisfy the associated constraints. As a result, the penalties provide a means of introducing cuts that force the AP solution to form a tour.

*Example 3.2*

Consider the travelling salesman graph whose cost matrix is given in Table 3.3. We will solve the TSP by using first the branch and bound scheme developed by Murty and then the proposed algorithm.

Table 3.3 - The cost matrix corresponding to the TSP solved in Example 3.2

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | ∞ | 4 | 10 | 18 | 5 | 10 |
| 2 | 4 | ∞ | 12 | 8 | 2 | 6 |
| 3 | 10 | 12 | ∞ | 4 | 18 | 16 |
| 4 | 18 | 8 | 4 | ∞ | 14 | 6 |
| 5 | 5 | 2 | 18 | 14 | ∞ | 16 |
| 6 | 10 | 6 | 16 | 6 | 16 | ∞ |

The minimum cost assignment solution to this problem and the resultant matrix is given in Table 3.4.

Table 3.4 - AP solution to the Example 3.2

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | ∞ | 0 | 3 | 14 | 0 | 1 |
| 2 | 0 | ∞ | 8 | 7 | 0 | 0 |
| 3 | 3 | 8 | ∞ | 0 | 13 | 7 |
| 4 | 14 | 7 | 0 | ∞ | 12 | 0 |
| 5 | 0 | 0 | 13 | 12 | ∞ | 9 |
| 6 | 1 | 0 | 7 | 0 | 9 | ∞ |

cost = 30

AP solution:

{(1,5),(2,6),(3,4),(4,3),
(5,1),(6,2)}

Once the AP solution is partitioned, the list at the end of the initial stage consists of five nodes as given in Table 3.5. Branching to

Table 3.5 - List of nodes at the end of the initial branching in Murty's algorithm

| Nodes: | AP solutions: |
|---|---|
| $S_1 = \{(\overline{1,5})\}$ | $\{(1,6),(2,5),(3,4),(4,3),(5,1),(6,2)\} = 31$ |
| $S_2 = \{(1,5),(\overline{2,6})\}$ | $\{(1,5),(2,1),(3,6),(4,3),(5,2),(6,4)\} = 37$ |
| $S_3 = \{(1,5),(2,6),(\overline{3,4})\}$ | $\{(1,5),(2,6),(3,1),(4,3),(5,2),(6,4)\} = 33$ |
| $S_4 = \{(1,5),(2,6),(3,4),(\overline{4,3})\}$ | $\{(1,5),(2,6),(3,4),'4,2),(5,1),(6,3)\} = 44$ |
| $S_5 = \{(1,5),(2,6),(3,4),(4,3),(\overline{5,1})\}$ | $\{(1,5),(2,6),(3,4),(4,3),(5,2),(6,1)\} = 31$ |

$S_1$, the least cost node, we obtain the partition listed in Table 3.6. We next branch to $S_5$ since, it possesses the least cost AP solution.

Table 3.6 - List of nodes at the end of the second branching in Murty's algorithm

| Nodes: | AP solutions: |
|---|---|
| $S_{11} = \{(\overline{1,5}),(\overline{1,6})\}$ | $\{(1,3),(2,5),(3,4),(4,6),(5,1),(6,2)\} = 33$ |
| $S_{12} = \{(\overline{1,5}),(1,6),(\overline{2,5})\}$ | $\{(1,6),(2,1),(3,4),(4,3),(5,2),(6,5)\} = 40$ |
| $S_{13} = \{(\overline{1,5}),(1,6),(2,5),(\overline{3,4})\}$ | $\{(1,6),(2,5),(3,1),(4,3),(5,2),(6,4)\} = 34$ |
| $S_{14} = \{(\overline{1,5}),(1,6),(2,5),(\underline{3,4}),(4,3)\}$ | $\{(1,6),(2,5),(3,4),(4,2),(5,1),(6,3)\} = 45$ |
| $S_{15} = \{(\overline{1,5}),(1,6),(2,5),(\underline{3,4}),(4,3),(\overline{5,1})\}$ | $\{(1,6),(2,5),(3,4),(4,3),(5,2),(6,1)\} = 32$ |

in rank. The corresponding partition consists of only one node in which the AP is infeasible. Choosing the next node to branch we see

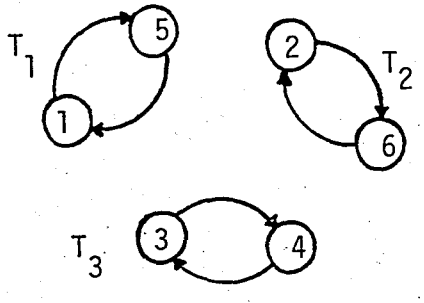
that $M_{15}$ has the least cost. However, this partition also consists of only one node with an infeasible AP solution (Table 3.7). As a result we, can either branch to $S_3$ or to $S_{11}$ since both nodes have the same

Table 3.7 - The third and the fourth partitions in Murty's algorithm

| Nodes: | AP solution: |
|---|---|
| $S_{51} = \{(1,5),(2,6)(3,4),(4,3),(\overline{5,1}),(\overline{5,2})\}$ | infeasible |
| $S_{151} = \{(\overline{1,5}),(1,6),(2,5),(3,4),(4,3),(\overline{5,1}),(\overline{5,2})\}$ | infeasible |

cost. Note that, the assignments corresponding to these nodes form two travelling salesman tours. Moreover, these tours are the same in the sense that they represent the same solution for the undirected graph. Thus, the optimum solution to the TSP is given by one of the AP solutions with a cost of 33. The number of nodes in the corresponding decision tree is 13 which means that the solution is obtained by solving 13 APs.

Now, let us apply the proposed algorithm to the problem. The solution to the AP and the corresponding cost matrix is obtained as given in Table 3.4. The associated subtours and the penalties are shown in Figure 3.4a. We start from the minimum penalty and solve




$\overline{p}_{15} = 0$ $\quad \overline{p}_{43} = 3$

$\overline{p}_{26} = 0$ $\quad \overline{p}_{51} = 0$

$\overline{p}_{34} = 3$ $\quad \overline{p}_{62} = 0$

$Z_o = 30$

Figure 3.4a - Subtours and penalties corresponding to the AP solution

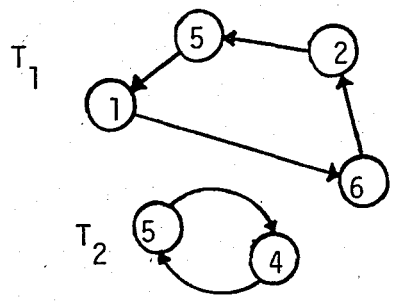
the corresponding AP. Then, the AP corresponding to the next minimum penalty is solved. The procedure is continued until the best solution found so far is less than the next penalty to be considered. The following solutions are obtained in each case:

| | Updates: | AP solution: | |
|---|---|---|---|
| (1) | $C_{1j} = \infty \quad \forall j \in T_1$ | {(1,6),(2,5),(3,4),(4,3),(5,1),(6,2)} | $Z_1 = 1$, $Z_1^* = 1$ |
| (2) | $C_{2j} = \infty \quad \forall j \in T_2$ | {(1,6),(2,5),(3,4),(4,3),(5,1),(6,2)} | $Z_1 = 1$, $Z_1^* = 1$ |
| (3) | $C_{5j} = \infty \quad \forall j \in T_1$ | {(1,5),(2,6),(3,4),'4,3),(5,2),(6,1)} | $Z_1 = 1$, $Z_1^* = 1$ |
| (4) | $C_{6j} = \infty \quad \forall j \in T_2$ | {(1,5),(2,6),(3,4),(4,3),(5,2),(6,1)} | $Z_1 = 1$, $Z_1^* = 1$ |

At this stage, since the best solution at hand is less than the next penalty to be considered and this solution does not form a tour, we set $Z_o = 31$ and continue with the cost matrix corresponding to the solution (1) (Table 3.8). The associated subtours and penalties are shown in Figure 3.4b.

Table 3.8 - Cost matrix corresponding to the solution (1)

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | ∞ | 0 | 2 | 14 | -1 | 0 |
| 2 | 0 | ∞ | 8 | 8 | 0 | 0 |
| 3 | 2 | 8 | ∞ | 0 | 12 | 6 |
| 4 | 14 | 8 | 0 | ∞ | 12 | 0 |
| 5 | 0 | 1 | 13 | 13 | ∞ | 9 |
| 6 | 0 | 0 | 6 | 0 | 8 | ∞ |




$\bar{p}_{16} = 2$ $\bar{p}_{43} = 2$

$\bar{p}_{25} = 7$ $\bar{p}_{51} = 13$

$\bar{p}_{34} = 2$ $\bar{p}_{62} = 0$

Figure 3.4b - Subtours and penalties corresponding to the AP solution at the end of the first stage.

The solutions corresponding to the successively solved APs are as follows:

| | Updates: | AP solution: | |
|---|---|---|---|
| (1) | $C_{6j} = \infty \quad \forall j \in T_1$ | {(1,6),(2,5),(3,1),(4,3),(5,2),(6,4)} | $Z_1 = 3,\ Z_1^* = 3$ |
| (2) | $C_{1j} = \infty \quad \forall j \in T_1$ | {(1,6),(2,5),(3,4),(4,6),(5,1),(6,4)} | $Z_1 = 2,\ Z_1^* = 2$ |

At this point, we do not need to continue with solving other AP's since $Z_1^* = 2$ is equal to the next penalty that would be considered. On the other hand, solution (2) is a tour and therefore, the best achievable solution to the TSP.

As can be seen, the number of APs that have been solved is 6 which constitutes a 50% reduction as compared with the previous algorithm. Another advantage of the algorithm is that each of the problems can be solved by storing the cost matrix of the previous problem from which it was derived. Thus, the storage requirements cause no problems even for large problems. After all, the efficiency of the algorithm is highly dependent on the structure of the cost matrix although the cut introduction procedure reduces the number of APs to be solved. However, since we always set one of the assignments to infinity the other (n-1) assignments are still valid. Therefore, the solution of a modified problem can be derived by reentering the Hungarian algorithm at the last step in order to increase the number of assignments from (n-1) to n and consequently, to produce the optimal

solution to the new AP with the least possible computation effort.

A disadvantage of the algorithm arises from the restriction imposed in calculating the penalties at the third step. That is, we are restricted to calculating positive penalties. In addition, the AP solved must end up with a positive objective function value. This value gives a magnitude of the improvement made from the present solution towards the optimum solution to the TSP. An AP solution with a negative value means that the overall solution is getting worse. This may lead to an infinite loop going back and forth in the solution space. Therefore, all the calculations have to be made in the positive domain. As a consequence of this fact, each time only one variable which has been removed from the basis previously, may enter the basis once again. It is not possible, that two or more variables which have become nonbasic enter the basis simultaneously. In that case, the optimal solution may not be caught.

Another case, which may end up with missing the optimal solution occurs when there exists an AP solution which has the same objective function value as the TSP solution. The algorithm may arrive at this solution but ignore it since it does not form a tour. As a result, a nonoptimal TSP solution may be obtained. After all, considering these drawbacks the algorithm may seem to be inefficient when compared with the other proposed methods. However, it represents an efficient extreme point ranking approach.

## 3.3 ALGORITHM III

For a given graph G = (N,E) with arc costs given by the matrix C the longest path problem is to find a simple path between two specified nodes such that the sum of the arc length is maximum provided that such a path exists and no positive cost circuit exists in G. If such a circuit exists, traversing the circuit an arbitrary large number of times will result in a path with an arbitrary large ($\to \infty$) cost so that the best path is not uniquely defined. If on the other hand, such circuits exist but are excluded from consideration somehow, then finding the longest path between two specified nodes becomes equivalent to the problem of finding the longest Hamiltonian path of the graph with the specified end nodes. As a matter of fact, if each entry $C_{ij}$ of the cost matrix C is subtracted from a large number, L, to produce a new cost matrix C' in which the triangle inequality is satisfied, then the longest path between any specified two nodes excluding positive circuits must pass through all other nodes. As a result, the following theorem due to Hardgrave and Nemhauser [55] allows one to solve the TSP as a longest path problem defined as above.

*Theorem 3.1.* Given the nodes {1,...,n}, arcs (i,j) and cost matrix C construct a new graph containing the nodes and arcs from the original graph plus one additional node denoted by $\alpha$ and an additional arc $(j,\alpha)$ for each j such that (j,1) is an arc in the original graph. The costs in the new graph are

$$\overline{C}_{ij} = 0 \qquad \forall i$$

$$\overline{C}_{j1} = -\infty \qquad \forall j \neq 1$$

$$\overline{C}_{j\alpha} = L - C_{j1} \qquad \forall j \neq \alpha$$
$$\overline{C}_{ij} = L - C_{ij} \qquad \text{otherwise}$$

where L is any finite number greater than the sum of n largest $C_{ij}$. Then, a longest path from 1 to $\alpha$ in the new graph contains every intermediate node $\{2,\ldots,n\}$ and if $\{1,i_1,i_2,\ldots,i_{n-1},\alpha\}$ is such a longest path, $\{1,i_1,\ldots,i_{n-1},1\}$ is an optimal tour.

Unfortunately, the theorem has not been useful since no efficient algorithms for the longest path problems defined as above have been discovered. However, the proposed algorithm provides a heuristic means of finding the longest path of a graph in which all of the nodes appear once and only once by the use of a dynamic programming type approach which is very similar to Ford's shortest path algorithm.

The proposed algorithm starts with subtracting each entry $C_{ij}$ of the cost matrix C from a large number L in order that the triangle inequality holds in the resultant matrix C', i.e. $C'_{ij} \leq C'_{ik} + C'_{kj}$. Eventually, the method is iteratively based on node labelling where at the end of the $k^{th}$ iteration the labels represent values on the longest path (from an arbitrarily chosen root node, s, to all other nodes) which contains (k+1) arcs. Once the lengths of the longest paths from s to all other nodes are obtained, the paths are identified immediately since another label representing predecessor nodes on the path is stored for each node during the computations. The algorithm can be summarized as follows:

Let $R(.)$ = the union of the reachible sets of nodes in $(.)$

$R^{-1}(.)$ = the union of the reaching sets of nodes in $(.)$

$\ell^{k+1}(i)$ = label on node i at the end of the $k^{th}$ iteration.

$\theta^{k+1}(i)$ = label showing the predecessor node of i in the longest path during the $k^{th}$ iteration.

$p^{k+1}(i)$ = the node set in the longest path from s to i at the end of the $k^{th}$ iteration.

0. Let $C'_{ij} = L - C_{ij}$ $\forall i,j \in N$ where L is a very large number

1. Set $S = R(s)$, $k = 1$, $L^1(s) = 0$, $L^1(i) = C'_{si}$ for all $i \in R(s)$ and $L(i) = \infty$ for all other i, $\theta^1(i) = s$ for all $i \in R(s)$, and $p^1(i) = \{s,i\}$ for all $i \in R(s)$.

2. For every node $i \in R(S)$, $i \neq s$, update its label according to the expression

$$\ell^{k+1}(i) = \max_{\substack{j \in T_i \\ i \notin p^k(j)}} \{\ell^k(j) + C'_{ji}\} = \ell^k(r) + C'_{ri}$$

where $T_i = (R^{-1}(i) \cap S)$. Set $p^{k+1}(i) = p^k(r) \cup \{r\}$, $\theta^{k+1}(i) = r$. For those nodes $i \notin R(S)$, set $\ell^{k+1}(i) = \ell^k(i)$, $p^{k+1}(i) = p^k(i)$ and $\theta^{k+1}(i) = i$.

3. If $k = n-2$, then stop. Find the longest paths from s to all other nodes i by tracing in the reverse order as

$$s,\theta^2(\theta^3(\ldots\theta^{n-1}(i))),\ldots,\theta^{n-2}(\theta^{n-1}(i)),\theta^{n-1}(i),i$$

Complete the longest paths for which arcs (i,s) exist to Hamiltonian circuits and choose the one with the maximum cost as the best achievable solution. Otherwise continue.

4. Update the set S as

$$S = \{i | \ell^{k+1}(i) \neq L^k(i)\}$$

5. Set $k = k+1$ and return to setp (2).

*Example 3.3*

Consider the undirected graph G whose cost matrix is given in Table 3.9. It is required to solve the TSP on this matrix. The algorithm proceeds as follows: Let L = 50. The matrix obtained

Table 3.9 - The cost matrix corresponding to the TSP solved in Example 3.3

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | ∞ | 9 | 8 | 7 | 6 | 10 |
| 2 | 9 | ∞ | 10 | 9 | 15 | 20 |
| 3 | 8 | 10 | ∞ | 5 | 15 | 25 |
| 4 | 7 | 9 | 5 | ∞ | 20 | 5 |
| 5 | 6 | 15 | 15 | 20 | ∞ | 20 |
| 6 | 10 | 20 | 25 | 5 | 20 | ∞ |

after subtracting each entry $C_{ij}$ from L is given in Table 3.10.

Table 3.10- The cost matrix after subtracting each entry from a large number L = 50

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | ∞ | 41 | 42 | 43 | 44 | 40 |
| 2 | 41 | ∞ | 40 | 41 | 35 | 30 |
| 3 | 42 | 40 | ∞ | 45 | 35 | 25 |
| 4 | 43 | 41 | 45 | ∞ | 30 | 45 |
| 5 | 44 | 35 | 35 | 30 | ∞ | 30 |
| 6 | 40 | 30 | 25 | 45 | 30 | ∞ |

Step (1) $s = 1$, $S = \{2,3,4,5,6\}$, $k = 1$

$\ell^1(2) = 41$ $\theta^1(2) = 1$ $p^1(2) = \{1,2\}$

$\ell^1(3) = 42$ $\theta^1(3) = 1$ $p^1(3) = \{1,3\}$

$\ell^1(4) = 43$ $\theta^1(4) = 1$ $p^1(4) = \{1,4\}$

$\ell^1(5) = 44$ $\theta^1(5) = 1$ $p^1(5) = \{1,5\}$

$\ell^1(6) = 40$ $\theta^1(6) = 1$ $p^1(6) = \{1,6\}$

Step (2) $R(S) = \{1,2,3,4,5,6\}$

$\ell^2(2) = 84$ $\theta^2(2) = 3$ $p^2(2) = \{1,2,3\}$

$\ell^2(3) = 88$ $\theta^2(3) = 4$ $p^2(3) = \{1,3,4\}$

$\ell^2(4) = 87$ $\theta^2(4) = 3$ $p^2(4) = \{1,3,4\}$

$\ell^2(5) = 77$ $\theta^2(5) = 3$ $p^2(5) = \{1,3,5\}$

$\ell^2(6) = 88$ $\theta^2(6) = 4$ $p^2(6) = \{1,4,6\}$

step (3) $k < 4$ , continue

step (4) $S = \{2,3,4,5,6\}$

step (5) $k = 2$

step (2) $R(S) = \{1,2,3,4,5,6\}$

$\ell^3(2) = 128$ $\theta^3(2) = 3$ $p^3(2) = \{1,2,3,4\}$

$\ell^3(3) = 113$ $\theta^3(3) = 6$ $p^3(3) = \{1,3,4,6\}$

$\ell^3(4) = 125$ $\theta^3(4) = 2$ $p^3(4) = \{1,2,3,4\}$

$\ell^3(5) = 123$ $\theta^3(5) = 3$ $p^3(5) = \{1,3,4,5\}$

$\ell^3(6) = 132$ $\theta^3(6) = 4$ $p^3(6) = \{1,3,4,6\}$

step (3) $k < 4$, continue

step (4) $S = \{2,3,4,5,6\}$

step (5) $k = 3$

step (2) $R(S) = \{1,2,3,4,5,6\}$

| | | |
|---|---|---|
| $\ell^4(2) = 162$ | $\theta^4(2) = 6$ | $p^4(2) = \{1,2,3,4,6\}$ |
| $\ell^4(3) = 113$ | $\theta^4(3) = 3$ | $p^4(3) = \{1,3,4,6\}$ |
| $\ell^4(4) = 125$ | $\theta^4(4) = 4$ | $p^4(4) = \{1,2,3,4\}$ |
| $\ell^4(5) = 163$ | $\theta^4(5) = 2$ | $p^4(5) = \{1,2,3,4,5\}$ |
| $\ell^4(6) = 170$ | $\theta^4(6) = 4$ | $p^4(6) = \{1,2,3,4,6\}$ |

step (3) $k < 4$, continue

step (4) $S = \{2,5,6\}$

step (5) $k = 4$

step (2) $R(S) = \{1,2,3,4,5,6\}$

| | | |
|---|---|---|
| $\ell^5(2) = 162$ | $\theta^5(2) = 2$ | $p^5(2) = \{1,2,3,4,6\}$ |
| $\ell^5(3) = 113$ | $\theta^5(3) = 3$ | $p^5(3) = \{1,3,4,6\}$ |
| $\ell^5(4) = 125$ | $\theta^5(4) = 4$ | $p^5(4) = \{1,2,3,4\}$ |
| $\ell^5(5) = 200$ | $\theta^5(5) = 6$ | $p^5(5) = \{1,2,3,4,5,6\}$ |
| $\ell^5(6) = 200$ | $\theta^5(6) = 5$ | $p^5(6) = \{1,2,3,4,5,6\}$ |

step (3) $k = 4$, stop.

Longest paths:

path 1: $\{1,4,3,2,5,6\}$ $\quad cost_1 = 200$

path 2: $\{1,3,2,4,6,5\}$ $\quad cost_2 = 200$

Since $cost_1 + C'_{61} = 240 < cost_2 + C'_{51} = 244$, the best achievable solution to the TSP is selected as (1-3-2-4-6-5-1) with an original cost of 56. However, the optimum solution to this problem is (1-5-2-3-4-6-1) with a cost of 51. As can be seen, the algorithm may not be able to find the optimum solution since it has a memoryless property in the sense that stage k is only dependent on stage (k-1) and previous stages have no effect. On the other hand, the algorithm is efficient considering that it requires on the order of $n^3$ operations for the case of a completely connected graph of n nodes.

## 3.4 ALGORITHM IV

The TSPs defined in Euclidean two space often have computational advantages. Once the nodes of a TSP are points in a two-dimensional space, the triangle inequality is satisfied. It is then possible to generate reasonable and sometimes optimal solutions to the problem by appealing to the geometric properties of the space. Examples of geometric approaches are described by Norback and Love [45] and Or [46].

Almost all algorithms falling into this category take advantage of the exploitable properties of the problem structure. The following theorems reveal the implication of such properties.

*Theorem 3.2.* If the cost matrix C represents Euclidean distances then the optimal tour does not intersect itself [56].

The theorem is obvious since any two intersecting arcs in the Euclidean space can be replaced by two nonintersecting arcs of a less total cost.

*Theorem 3.3.* If H is the convex hull of the points representing the TSP in a two-dimensional Euclidean space, then the order in which the nodes appear on H is the same as the order in which they appear in the optimal tour [57].

This theorem is a direct consequence of Theorem 3.2. As a result if k nodes lie on a convex hull, then Theorem 3.3 reduces the total number of tours which are to be investigated from $(1/2)(n-1)!$ to $(n-1)!/(m-1)!$ (for undirected problems) where n is the total number of nodes associated with the problem.

Once the nodes on the convex hull are specified, the problem is to decide how to sequence the remaining interior nodes between different consecutive pair of nodes on the partial tour. The decision is made by considering the heights of the triangles whose bases are the arcs through consecutive pair of nodes in the partial tour and whose third vertices are the remaining interior nodes which have not been considered yet. Each time, a new partial tour having an additional node is constructed. The process of calculating the heights of triangles and choosing the appropriate one can then be repeated by using the new partial tour and the remaining interior nodes.

Assuming that the nodes are located in a two dimensional space we are sure that the triangle inequality is always satisfied. However, this may not be the case, when the cost matrix of a TSP contains arbitrarily chosen numbers. In order to achieve triangle inequality, all the elements $C_{ij}$ of the cost matrix are subtracted from a large number L. This may also be done when the cost matrix satisfies the triangle inequality since any matrix transformed in this manner does satisfy the

triangle inequality after the subtraction process. Obviously, the problem becomes to find a Hamiltonian circuit with the maximum length. Therefore, triangles with maximum heights are considered first in building the travelling salesman tour.

In order to begin the process of calculating the heights and choosing the largest one, the convex hull must be determined. In fact, if the nodes are located in a two-dimensional space and can be mapped on a paper, then the convex hull can be determined easily by taking a look at the whole layout. However, for problems which involve a very large number of nodes the procedure may not be that easy. Besides, it is not possible to determine a convex hull for problems which are not defined in Euclidean space. The following procedure due to Norback and Love [45] is applicable to problems in which the nodes have known coordinates:

1. Choose the node with the x coordinate of least value. This node is on the convex hull and can be labelled $h_1$ - the first hull node.
2. Using this node as vertex, form all possible angles whose sides are rays containing this node and another node of the problem. Choose the largest of these angles.
3. Choose one of the nodes that determine this angle other than the vertex and label it $h_2$, the second hull node.
4. Using $h_2$ as a vertex and the ray containing $h_1$ and $h_2$ as a side determine all angles whose remaining side contains $h_2$ and another node of the problem. Choose the largest of these angles and label the corresponding node $h_3$.

5. Repeat step (4) as many times as necessary, with the most recent hull node generated $h_i$, as vertex and the one given side of the angle ray containing $h_i$ and $h_{i-1}$.
6. The convex hull will be determined when the next candidate for a node on the hull is $h_1$.

Once the convex hull is given as an input, the proposed algorithm initiates a list keeping procedure. The list contains the necessary information about the existing arcs on the convex hull. All the book-keeping and manipulations are made on this list. The algorithm can be summarized as follows:

1. Find the convex hull H associated with the problem. Let T be the initial partial tour (T ≡ H if H can be found). Set $C'_{ij} = L - C_{ij}$ for all $i,j \in N$.
2. If T covers n nodes, then stop; T is the best achievable solution to the TSP. Otherwise continue.
3. For each arc in T, form a list by performing the following steps:
   a) Specify the end nodes of the arc $(i,j) \in T$.
   b) For each node $k \notin T$ calculate the height of the triangle determined by nodes, i, k, and j as

$$h = 2\sqrt{u(u - C'_{ik})(u - C'_{kj})(u - C'_{ij})}/C'_{ij}$$

   where $u = (C'_{ik} + C'_{kj} + C'_{ij})/2$.

c) Select the node corresponding to the largest height and record it as a candidate for being inserted between i and j in T.

4. Choose the arc (i,j) which has a candidate, k, corresponding to the maximum height in the list. Delete arc (i,j) from the list. Instead, include arcs (i,k) and (k,j) as the new arcs in the new partial tour T. Return back to step (2).

*Example 3.4*

Consider a complete undirected graph whose nodes are located in a two dimensional space as shown in Figure 3.5 and whose cost (distance) matrix is given in Table 3.11.

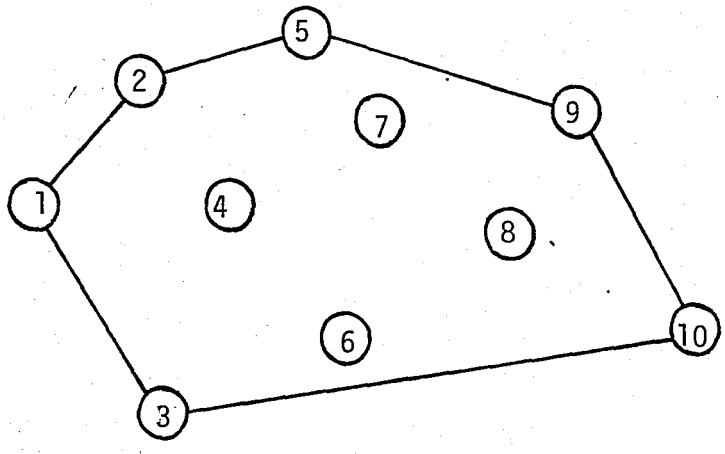



Figure 3.5 - The convex hull corresponding to the travelling salesman graph in Example 3.4.

The convex hull of this problem can be identified by taking a look at the node locations in Figure 3.5. Eventually, the convex hull is (1-2-5-9-10-3-1). Initially, we set T = (1-2-5-9-10-3-1) and subtract all the elements $C_{ij}$ from a large number L which is chosen to be 400 in this case. The resultant matrix is given in Table 3.12.

Table 3.11- The cost matrix of Example 3.4

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ∞ | 42 | 72 | 50 | 86 | 89 | 92 | 130 | 151 | 182 |
| 2 | 42 | ∞ | 91 | 36 | 45 | 86 | 61 | 108 | 120 | 162 |
| 3 | 72 | 91 | ∞ | 61 | 114 | 45 | 94 | 103 | 136 | 143 |
| 4 | 50 | 36 | 61 | ∞ | 54 | 50 | 45 | 81 | 102 | 133 |
| 5 | 86 | 45 | 114 | 54 | ∞ | 91 | 36 | 85 | 85 | 136 |
| 6 | 89 | 86 | 45 | 50 | 91 | ∞ | 61 | 58 | 92 | 101 |
| 7 | 92 | 61 | 94 | 45 | 36 | 61 | ∞ | 50 | 60 | 103 |
| 8 | 130 | 108 | 103 | 81 | 85 | 58 | 50 | ∞ | 36 | 54 |
| 9 | 151 | 120 | 136 | 102 | 85 | 92 | 60 | 36 | ∞ | 58 |
| 10 | 182 | 162 | 143 | 133 | 136 | 101 | 103 | 54 | 58 | ∞ |

Table 3.12- The cost matrix after subtracting each element from a large number L = 400

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ∞ | 358 | 328 | 350 | 314 | 311 | 308 | 270 | 249 | 218 |
| 2 | 358 | ∞ | 309 | 364 | 355 | 314 | 339 | 292 | 280 | 238 |
| 3 | 328 | 309 | ∞ | 339 | 286 | 355 | 306 | 297 | 264 | 257 |
| 4 | 350 | 364 | 339 | ∞ | 346 | 350 | 355 | 319 | 298 | 267 |
| 5 | 314 | 355 | 286 | 346 | ∞ | 309 | 364 | 315 | 315 | 264 |
| 6 | 311 | 314 | 355 | 350 | 309 | ∞ | 339 | 342 | 308 | 299 |
| 7 | 308 | 339 | 306 | 355 | 364 | 339 | ∞ | 350 | 340 | 297 |
| 8 | 270 | 292 | 297 | 319 | 315 | 342 | 350 | ∞ | 364 | 346 |
| 9 | 249 | 280 | 264 | 298 | 315 | 308 | 340 | 364 | ∞ | 342 |
| 10 | 218 | 238 | 257 | 267 | 264 | 299 | 297 | 346 | 342 | ∞ |

Note that the partial tour T does not cover n nodes and we have four remaining interior nodes {4,6,7,8}. As a result, the list formed in the third step of the algorithm is given in Table 3.13. Since the

Table 3.13 - List for arcs in T in the first step

| Starting node | Ending node | Candidate node | Height |
|---|---|---|---|
| 1 | 2 | 4 | 308.60 |
| 2 | 5 | 4 | 307.04 |
| 5 | 9 | 7 | 313.88* |
| 9 | 10 | 8 | 310.67 |
| 10 | 3 | 6 | 293.47 |
| 3 | 1 | 4 | 302.78 |

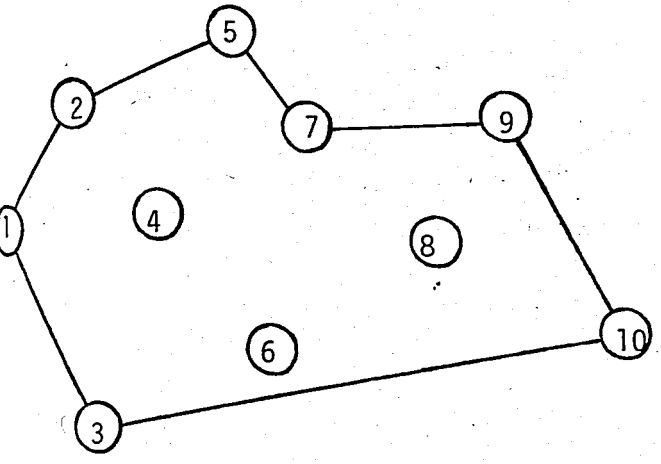
maximum height in the list corresponds to arc (5,9) we insert node 7 between nodes 5 and 9 and obtain the new tour as T = (1-2-5-7-9-10-3-1). The tour is shown in Figure 3.6a. Now, T covers seven nodes and the set of the remaining nodes is {4,6,8}. Therefore, the list is updated as given in Table 3.14. The maximum height in this list corresponds to the arc (7,9). This implies the insertion of node 8 between nodes 7 and 9 and result in the partial tour shown in Figure 3.6b. Once again, T does not cover n nodes and the remaining node set {4,6} consists of two nodes. The list is updated as shown in Table 3.15. In this case the maximum height corresponds to the arc (1,2) with a value of 308.60. After node 4 is inserted between nodes 1 and 2 the resultant tour (1-4-2-5-7-8-9-10-3-1) is as shown in Figure 3.6c. At this stage node 6 remains to be inserted

Table 3.14 - List for arcs in T in the second step

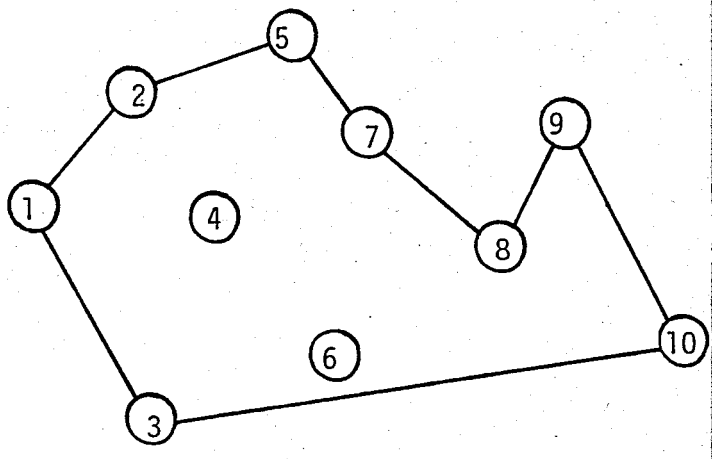
| Starting node | Ending node | Candidate node | Height |
|---|---|---|---|
| 1 | 2 | 4 | 308.60 |
| 2 | 5 | 4 | 307.40 |
| 5 | 7 | 4 | 299.45 |
| 7 | 9 | 8 | 313.66 |
| 9 | 10 | 8 | 310.62* |
| 10 | 3 | 6 | 293.47 |
| 3 | 1 | 4 | 302.72 |

Table 3.15 - List for arcs in T in the third step

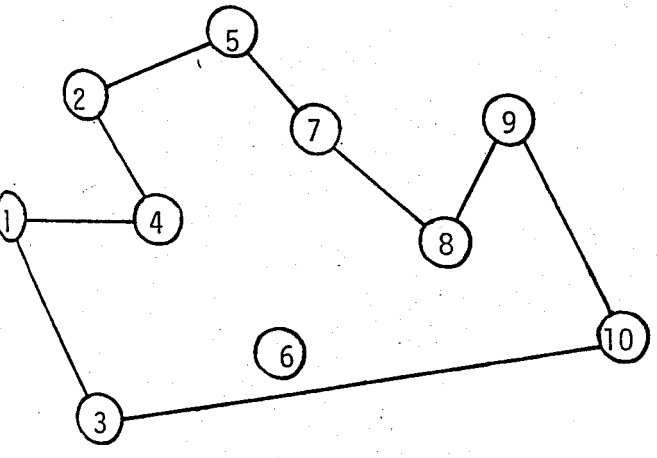
| Starting node | Ending node | Candidate node | Height |
|---|---|---|---|
| 1 | 2 | 4 | 308.60* |
| 2 | 5 | 4 | 307.04 |
| 5 | 7 | 4 | 299.45 |
| 7 | 8 | 6 | 292.08 |
| 8 | 9 | 6 | 268.08 |
| 9 | 10 | 6 | 250.65 |
| 10 | 3 | 6 | 293.47 |
| 3 | 1 | 4 | 302.72 |



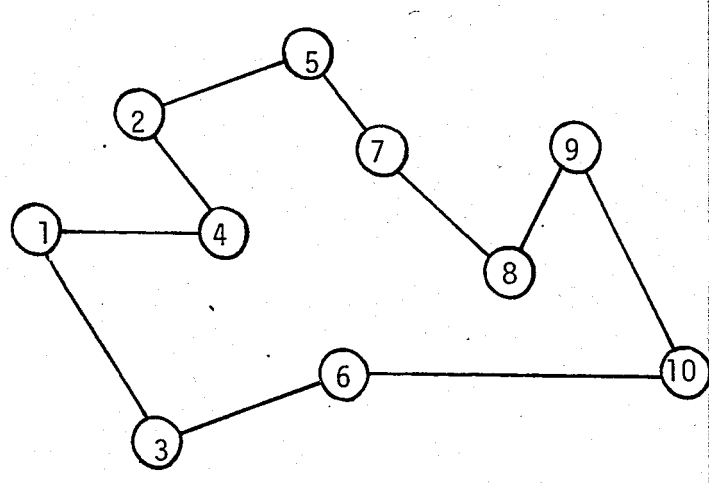

(a)




(b)




(c)




(d)

Figure 3.6 - Steps in building the travelling salesman tour

in order to obtain the final tour. The corresponding list is given in Table 3.16. The maximum height is this list corresponds to the arc (10,3). Inserting node 6 between nodes 10 and 3, the best

Table 3.16 - List for arcs in T in the fourth step

| Starting node | Ending node | Candidate node | Height |
|---|---|---|---|
| 1 | 4 | 6 | 278.62 |
| 4 | 2 | 6 | 276.31 |
| 2 | 5 | 6 | 255.96 |
| 5 | 7 | 6 | 267.14 |
| 7 | 8 | 6 | 292.08 |
| 8 | 9 | 6 | 268.08 |
| 9 | 10 | 6 | 250.65 |
| 10 | 3 | 6 | 293.47* |
| 3 | 1 | 6 | 287.20 |

achievable tour is found to be (1-4-2-5-7-8-9-10-6-3-1) with an original total cost of 529 (Figure 3.6d).

Comparing the proposed measure used to determine the sequence of nodes with other measures proposed before we see that once the heights are used the algorithm does not fail in cases where other measures have been observed to fail. For instance, consider the example in which the largest angle method proposed by Norback and Love fails. Using the largest angle method, the tour generated is

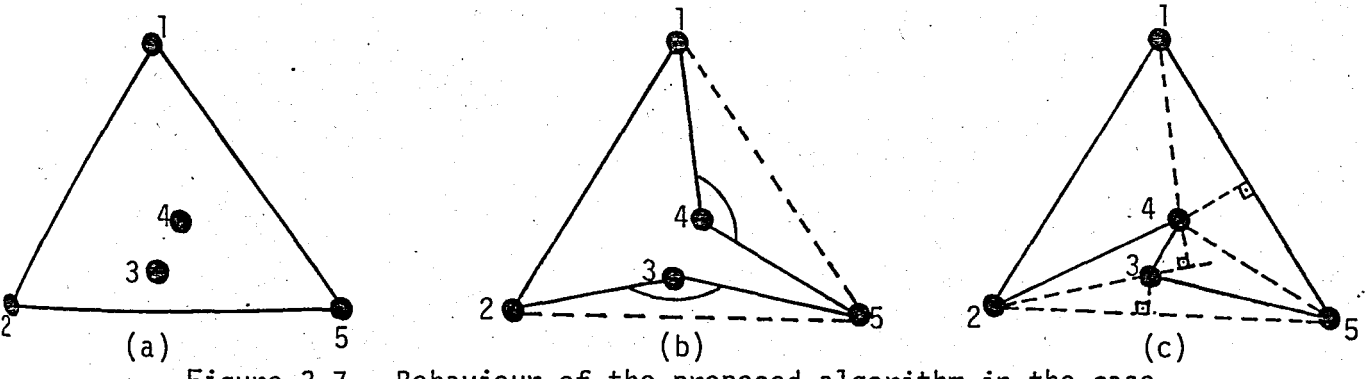



Figure 3.7 - Behaviour of the proposed algorithm in the case where Norback's and Love's largest angle method fails.

given in Figure 3.7b whereas applying the heights as a measure yields the tour given in Figure 3.7c. Note that, in this case the minimum heights are chosen since the problem is defined in the Euclidean space and the triangle inequality is satisfied without need of subtracting the elements of the cost matrix from a large number.

Similarly the eccentric ellipse method fails in the example shown in Figure 3.8. The choice mechanism in the eccentric ellipse method will sequence node 5 between nodes 2 and 4 as shown in Figure 3.8b. However, when heights of the triangles are applied a less costly tour can be obtained as shown in Figure 3.8c.

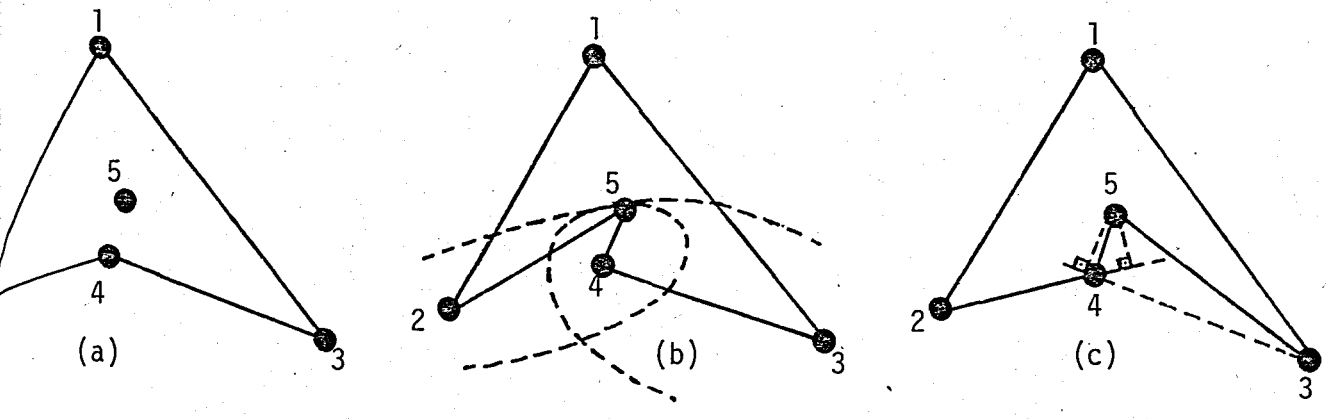



Figure 3.8 - Behaviour of the proposed algorithm in the case where Norback's and Love's eccentric ellipse method fails.

As it has been reported by Or [46] the measures used in his algorithm namely (i) DIST = $C_{ik} + C_{kj} - C_{ij}$ (ii) RATIO = $(C_{ik}-C_{kj})/C_{ij}$ fail since they do not apply any preference when ties have to be broken. In addition consider the case shown in Figure 3.9 for the DIST measure. The minimum DIST is given by $C_{24} + C_{41} - C_{21}$ whereas the minimum height measure inserts node 4 between nodes 2 and 3 which in case yields a less costly solution. In addition to these advantages it is less

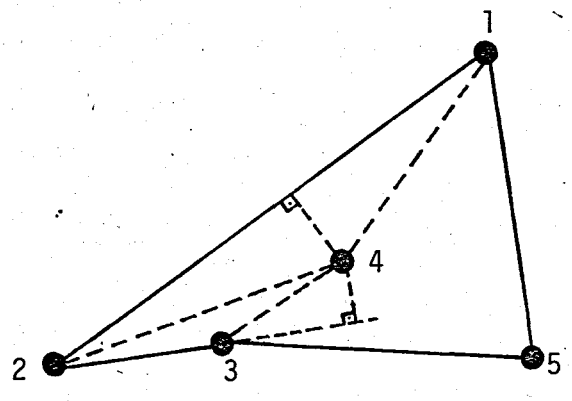



Figure 3.9 - Comparison of the height criterion with other criteria

probable that two or more heights happen to be equal and ties have to be broken. However, it is difficult to make any comparison with the third criterion proposed by Or which is MULT = DIST x RATIO. After all, our algorithm seems to be more advantageous in any case. Note that, MULT requires two operations, i.e., the calculation of DIST and the calculation of RATIO as compared with out algorithm.

The efficiency of the algorithm is highly dependent on the topological conditions of the problem. For problems defined in the Euclidean space the solutions obtained by starting with a convex hull are better than the solutions obtained by starting with an arbitrary partial tour. Note that, although the convex property of the convex hull is lost immediately after inserting a node into it, all the other nodes remain interior with respect to the new boundaries. This structure, however, plays an important role in obtaining better solutions than an arbitrary partial tour would yield.

A general advantage of the algorithm is that it is easy to apply. Reasonable solutions to problems where the number of nodes is small enough can be found quickly without the aid of a computer but just with the help of a pocket calculator.

## 3.5 COMPUTATIONAL RESULTS

Since the published computational results of various algorithms found in literature are given for different problems solved by different computers comparisons based on these results are difficult. In terms of computational effort we had two alternatives to choose. We could either base the comparisons on the published results which were tested in different environments or write all the programs in order to test them in our environment. In the former case, we could have been unfair to algorithms tested under different conditions. The latter case was infeasible in the sense that it would have been far beyond the scope of this study. Therefore, neither of the alternatives were chosen. On the other hand, we were unable to find the data of the problems tested by other authors since they are unpublished. As a result, the computational aspects were examined on the relative merits of the proposed algorithms.

To check the effectiveness of the proposed methods in solving the TSP 35 complete Euclidean problems were generated with points randomly selected from the unit square ($\{(x,y)|0 \leq x \leq 1, 0 \leq y \leq 1\}$). The problems contained nodes ranging between 10 and 70. The optimum solutions to the problems with at most 20 nodes were found by using Little's branch and bound approach. The methods were then applied

to the same problems. For the problems which contain more than 20 nodes, only comparisons between the proposed algorithms are made since an inordinate amount of time is required to solve problems of that size by using Little's branch and bound approach.

There are several different interrelated measures to consider in order to define the power of each heuristic method separately. For instance consider the reductions applied in the first method. The effect of reduction is felt in several ways. The most obvious is a considerable decrease in the number of arcs on which the Hamiltonian circuit search takes place. Results reveal that after connectedness and unilateral connectedness is achieved, the number of arcs that comprise the subgraph is less than about 10% of the total number of graphs existing in the original graph (Table 3.17). As a result, the computation effort is decreased. However, this saving in effort is not the only effect in introducing reduction. Rather, reduction introduces a bias into the procedure when no Hamiltonian circuit is formed although the necessary conditions are satisfied. In that case, Little's algorithm is applied partially to the resultant matrix. The solutions obtained by using the resultant matrix are different than the ones that can be obtained by using the original matrix. Results, regarding this fact are given in Table 3.18. Another fact that has been observed is that as the number of nodes increases, the probability that a Hamiltonian circuit will be produced in the subgraph decreases. Moreover, the solutions obtained tend to be only suboptimal solutions. In other words the application of Little's algorithm partially may not be able to break out of this to actually get the optimum solution. The results are indicated in Table 3.16a and Table 3.16b.

Table 3.17 - Computational results regarding Algorithm I

| n | Total No. of arcs | No. of arcs in $G_o$ | Solution in $G_o$ | n | Total No. of arcs | No. of arcs in $G_o$ | Solution in $G_o$ |
|---|---|---|---|---|---|---|---|
| 10 | 90 | 28 | Yes | 40 | 1560 | 112 | No |
| 10 | 90 | 32 | No | 40 | 1560 | 114 | No |
| 10 | 90 | 32 | Yes | 50 | 2450 | 154 | No |
| 10 | 90 | 25 | Yes | 50 | 2450 | 134 | No |
| 10 | 90 | 26 | No | 50 | 2450 | 150 | No |
| 20 | 380 | 52 | No | 50 | 2450 | 134 | No |
| 20 | 380 | 52 | No | 50 | 2450 | 138 | No |
| 20 | 380 | 68 | Yes | 60 | 3540 | 178 | No |
| 20 | 380 | 62 | Yes | 60 | 3540 | 182 | No |
| 20 | 380 | 50 | No | 60 | 3540 | 174 | No |
| 30 | 870 | 82 | No | 60 | 3540 | 160 | No |
| 30 | 870 | 92 | No | 60 | 3540 | 193 | No |
| 30 | 870 | 84 | No | 70 | 4830 | 206 | No |
| 30 | 870 | 86 | No | 70 | 4830 | 198 | No |
| 30 | 870 | 82 | No | 70 | 4830 | 198 | No |
| 40 | 1560 | 115 | No | 70 | 4830 | 192 | No |
| 40 | 1560 | 112 | No | 70 | 4830 | 212 | No |
| 40 | 1560 | 106 | No | | | | |

Table 3.18 - Results regarding the application of Little's algorithm partially to the reduced matrix or to the original matrix

| Problem size | Solution found by using the reduced matrix | Solution found by using the original matrix |
|---|---|---|
| 10 | 362 | 305 |
| 10 | 279 | 279 |
| 20 | 381 | 386 |
| 20 | 462 | 433 |
| 20 | 488 | 475 |

Considering the computational complexity of this algorithm it can be seen that the number of operations made during the execution is of order $n^3$ where n is the number of nodes in the graph. The algorithm consists of three parts: (strong) connectedness of the subgraph can be achieved in n (n-2) operations. This results from the fact that, in the worst case, the first reduction may end with $\lfloor n/2 \rfloor$ subtours where $\lfloor \cdot \rfloor$ indicates the integer part of (.). Thus, at most, (n-2) other reductions are needed to achieve (strong) connectedness. On the other hand, since n operations are needed to define the reachible set of a node, the total number of operations increases to $n^2$ when all nodes are considered. As a result of repeating the checking procedure (n-2) times in the worst case the total number of operations is of order $O(n^3)$. Similarly, in the second part of the algorithm, the number of operations needed for checking unilateral connectedness is $n(n-1)^2$, since (n-1) operations are needed to find the reachible sets of the (n-1) distinct nodes after a specific node is removed from the

Table 3.19a - Computational results for the proposed algorithms when applied to problems where $10 \leq n \leq 30$

| n | Optimum cost | Algorithm I | | Algorithm II | | Algorithm III | | Algorithm IV | |
|---|---|---|---|---|---|---|---|---|---|
| | | Cost | CPU* | Cost | CPU | Cost | CPU | Cost | CPU |
| 10 | 280 | 280 | 1.315 | 305 | 5.101 | 291 | 0.320 | 363 | 0.478 |
| 10 | 299 | 362 | 1.790 | 342 | 8.199 | 299 | 0.317 | 356 | 0.278 |
| 10 | 275 | 279 | 1.580 | 279 | 4.029 | 292 | 0.321 | 335 | 0.322 |
| 10 | 285 | 285 | 1.239 | 295 | 2.895 | 285 | 0.303 | 297 | 0.307 |
| 10 | 279 | 279 | 1.443 | 280 | 3.091 | 291 | 0.311 | 300 | 0.338 |
| 20 | 354 | 381 | 8.305 | 410 | 72.450 | 384 | 1.323 | 394 | 1.239 |
| 20 | 400 | 462 | 8.587 | 437 | 33.571 | 492 | 1.305 | 463 | 1.323 |
| 20 | 377 | 387 | 8.732 | 411 | 95.632 | 441 | 1.377 | 387 | 1.479 |
| 20 | 389 | 389 | 9.070 | 411 | 51.369 | 482 | 1.386 | 503 | 1.254 |
| 20 | 378 | 488 | 7.614 | 387 | 32.95 | 432 | 1.476 | 560 | 1.400 |
| 30 | - | 491 | 27.594 | - | - | 624 | 3.708 | 542 | 2.647 |
| 30 | - | 614 | 29.742 | - | - | 564 | 3.728 | 604 | 3.059 |
| 30 | - | 528 | 29.174 | - | - | 490 | 3.709 | 661 | 2.789 |
| 30 | - | 595 | 32.328 | - | - | 509 | 3.428 | 629 | 3.008 |
| 30 | - | 471 | 31.332 | - | - | 464 | 3.737 | 558 | 3.135 |

*CPU times are in seconds of UNIVAC 1106 computer, Boğaziçi University.

subgraph. As a result of repeating the process n times (i.e. each node in the subgraph is removed one by one) the number of operations needed for the whole procedure is of order $O(n^3)$. Finally, since Little's algorithm is applied until a feasible solution is found and therefore the order of this procedure is far from being greater than the orders of the other parts of the algorithm, the overall order of the proposed algorithm is $O(n^3)$. Note that, a careful examination of Table 3.19a and Table 3.19b reveals that the CPU times can be expressed approximately as $(n^3/1000)$.

The experiments made on the second method showed that the method might end up with high computation times as a result of jumping over the optimal solution. In other words, the algorithm may omit the optimal solution and then continue with the search in an unknown direction until a feasible solution is obtained. This may be the consequence of the fact that there may exist more than one AP solution with the same objective function value such that one is the optimum solution to the TSP and the other is not. On the other hand, the optimum solution may be omitted due to the fact that two previously basic but currently non-basic variables may not enter the basis at the same time. Hence, the algorithm may require a considerable amount of computation time in order to find at least a feasible tour. Experiments for this method were conducted upto 20 nodes. The results can be seen in Table 3.19a. It should be noted that the CPU times are still efficient as compared with the relevant ranking and subtour breaking methods in literature. In addition, problems of the same size showed a considerable variation in the computation times. This can be explained by the variation in

Table 3.19b - Computational results for the proposed algorithms when applied to problems where $40 \leq n \leq 70$

| n | Algorithm I | | Algorithm III | | Algorithm IV | |
|---|---|---|---|---|---|---|
| | Cost | CPU | Cost | CPU | Cost | CPU |
| 40 | 569 | 71.070 | 645 | 7.391 | 664 | 5.005 |
| 40 | 617 | 72.314 | 742 | 7.907 | 807 | 6.321 |
| 40 | 557 | 65.438 | 597 | 7.136 | 609 | 5.203 |
| 40 | 617 | 68.765 | 714 | 7.713 | 626 | 5.124 |
| 40 | 599 | 65.321 | 672 | 7.319 | 722 | 4.892 |
| 50 | 691 | 146.737 | 714 | 14.587 | 605 | 12.135 |
| 50 | 702 | 135.878 | 733 | 13.546 | 721 | 8.800 |
| 50 | 838 | 142.656 | 706 | 13.631 | 758 | 6.831 |
| 50 | 761 | 132.196 | 724 | 13.323 | 814 | 8.507 |
| 50 | 683 | 113.126 | 791 | 14.442 | 770 | 8.513 |
| 60 | 695 | 206.103 | 848 | 22.455 | 934 | 12.255 |
| 60 | 816 | 212.328 | 776 | 22.934 | 999 | 11.450 |
| 60 | 761 | 192.707 | 821 | 24.567 | 958 | 13.131 |
| 60 | 739 | 195.737 | 784 | 23.757 | 825 | 15.855 |
| 60 | 779 | 231.319 | 835 | 23.160 | 838 | 11.189 |
| 70 | 1010 | 328.886 | 967 | 44.765 | 1035 | 20.699 |
| 70 | 745 | 335.943 | 889 | 35.342 | 965 | 21.743 |
| 70 | 809 | 350.958 | 877 | 35.288 | 812 | 13.864 |
| 70 | 746 | 313.229 | 891 | 33.965 | 794 | 21.440 |
| 70 | 942 | 363.803 | 876 | 33.472 | 1015 | 17.355 |

the number of extreme points found between the optimum AP solution and the TSP solution.

Experimental results regarding the computation times of the third algorithm are also listed in Table 3.19a and Table 3.19b. The results indicate that this is a highly effective procedure for building a tour. The computation time required to solve problems with 70 nodes is about 35 seconds. The procedure was also capable of finding the optimal solutions in some of the problems. Of those that are not optimal, the deviation from the optimal value is less than 8%. But, of the 10 runs whose optimal solutions are known only 2 are optimal.

The computation effort of the algorithm can be expressed as follows: At the first stage, (n-2) operations are made for each of the (n-1) nodes other than the root node. At the second stage, the number of operations is (n-3) since (n-3) nodes remain to be sequenced. The other stages proceed similarly. Therefore, the total number of operations can be given by the expression

$$(n-1)\sum_{i=1}^{n-2} i = (n-1)^2(n-2)/2$$

which shows that the proposed algorithm is of order $O(n^3)$.

In terms of the computation effort, the last algorithm using the geometric approach is the most efficient one although it seems to find solutions worse than the others. This is because arbitrary convex hulls with the least possible number of nodes were input to the algorithm in order to measure its computational efficiency.

Consider that in the worst case the algorithm starts with a partial tour containing only two nodes and therefore two arcs. As a

result, the number of operations in the first step is 2(n-2) since there are two arcs and (n-2) remaining nodes to consider. Similarly, 3(n-3) operations are conducted for the second step. Overall,

$$\sum_{i=2}^{n-1} i(n - i)$$

operations are needed for the whole procedure. Expanding this expression we obtain

$$n \sum_{i=2}^{n-1} i - \sum_{i=2}^{n-1} i^2$$

which makes $(n^3 - 19n + 6)/6$ operations at most. Therefore we can conclude that the algorithm is of order $O(n^3)$. Analysing the results indicated in Table 3.19a and Table 3.19b we see that, even for the worst case, the computational effort of this algorithm is the best as compared with the others. Nevertheless, results regarding the costs are promising considering that the convex hulls which are necessary as input to the algorithm were not identified.

# IV. THE MULTI-DEPORT VEHICLE ROUTING PROBLEM AND ITS FORMULATION AS A TRAVELLING SALESMAN PROBLEM

## 4.1 INTRODUCTION

Vehicle routing and scheduling problems which involve the periodic collection and delivery of goods and services are both of theoretical and practical importance. The ideas lying under this subject have proven to be interesting for the researchers who are specialized in computer science and graph theory as well as operations research. On the other hand, routing and scheduling procedures contribute to saving a considerable amount of money by increasing the productivity, improving the operations, aiding in long range planning, handling the job scheduling and sequencing problems and controlling vehicle utilization from the financial point of view.

The vehicle routing problem (VRP) involves the designation of a set of routes which are sequences of pickup and/or delivery points that are to be traversed by vehicles in order, starting and ending at some depots. The problem is referred to as scheduling problem (VSP) when arrival and departure times of the vehicles are specified. As a matter of fact, the problem can be viewed as a

combined routing and scheduling problem when both routing and scheduling functions need to be performed.

A specific vehicle routing and/or scheduling problem can be described on the basis of a number of characteristics. The following taxonomy given by Bodin and Golden [58] is useful in identifying the type of the vehicle routing and/or scheduling problem that is being confined:

A. time to service a particular node or arc

   1. time specified and fixed in advance (pure VSP)
   2. time windows (combined VRP and VSP)
   3. time unspecified (VRP)

B. number of depots

   1. one depot
   2. more than one depot

C. size of fleet available

   1. one vehicle
   2. more than one vehicle

D. type of fleet available

   1. homogeneous case (all vehicles are the same)
   2. heterogeneous case (not all vehicles are the same)

E. nature of demands

   1. deterministic
   2. stochastic

F. location of demands

1. at nodes (not necessarily all)
2. on arcs (not necessarily all)
3. mixed

G. underlying graph

1. undirected
2. directed
3. mixed

H. vehicle capacity constraints

1. imposed - all the same
2. imposed - not all the same
3. not imposed

I. maximum vehicle route-times

1. imposed - all the same
2. imposed - not all the same
3. not imposed

J. costs

1. variable or routing costs
2. fixed operating or vehicle acquisition costs (capital costs)

K. operations

1. pickups only
2. delivery only
3. mixed

L. objective

1. minimize routing costs incurred
2. minimize sum of fixed and variable costs
3. minimize number of vehicles required

M. other (problem-dependent) constraints

Note that, this framework includes a vast variety of combinations which cover all of the well known problems as well as problems that have not received much research attention.

## 4.2 VEHICLE ROUTING PROBLEMS AS EXTENSIONS OF THE TRAVELLING SALESMAN PROBLEM

### 4.2.1 The Multiple Travelling Salesman Problem (MTSP)

Most of the VRPs are variants or extensions of the TSP. Actually, the problem of satisfying the demand at each node of a graph with a single vehicle of unlimited capacity while the total routing cost is being minimized is the TSP. Building upon the TSP, other problems progressing from the very simple to the more complex have been extended and synthesized. One of such problems is the MTSP which represents a large number of real world problems.

Given m salesmen and n nodes in a graph, the MTSP is to assign a subtour to each salesman such that the subtours start and end at a central depot and the sum of m subtour costs is minimized. The integer programming formulation of the MTSP can be obtained by changing the formulation of the TSP slightly [4] as

$$\text{minimize} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} C_{ij} x_{ij} \tag{4.1}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} x_{ij} = b_j = \begin{cases} m & \text{if } j = p \\ 1 & \text{otherwise} \end{cases} \qquad j = 1,\ldots,n \tag{4.2}$$

$$\sum_{j=1}^{n} x_{ij} = a_i = \begin{cases} m & \text{if } i = p \\ 1 & \text{otherwise} \end{cases} \qquad i = 1,\ldots,n \tag{4.3}$$

$$x_{ij} \in S \tag{4.4}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \tag{4.5}$$

where p is the node representing the central depot

S is the set of constraints prohibiting subtour solutions and can be represented by one of the constraint sets (2.4a),(2.4b) and (2.4c).

### 4.2.2 The Multi-Depot Vehicle Routing Problem (MDVRP)

The MDVRP is an extension of the MTSP which incorporates multiple depots. The MDVRP allows vehicles to reside at more than one depot and seeks for the minimum number of vehicles needed to satisfy all the demands while the total traversing cost is being minimized. The problem can be classified as being a pure VRP with more than one depot and more than one vehicle. The type of the fleet is assumed to be homogeneous, i.e. all of the vehicles are the same. The demands are deterministic. Neither vehicle capacity constraints nor maximum vehicle route times are imposed. That is, the vehicles are assumed to have capacities which exceed the total demand. The underlying graph can be undirected, directed or mixed. The problem can be related to delivery or pickup operations where only routing costs are being considered.

The integer programming formulation of the MDVRP can be summarized as follows:

Let the nodes of the graph be numbered such that the nodes 1,...,p denote the depots and the nodes p+1,...,p+n denote the demand points. Then, the formulation can be given as

$$\text{minimize} \quad \sum_{i=1}^{p+n} \sum_{j=1}^{p+n} \sum_{k=1}^{m} C_{ij} x_{ij}^{k} \tag{4.6}$$

subject to

$$\sum_{i=1}^{p+n} \sum_{k=1}^{m} x_{ij}^{k} = 1 \qquad j = p+1,\ldots,p+n \tag{4.7}$$

$$\sum_{j=1}^{p+n} \sum_{k=1}^{m} x_{ij}^{k} = 1 \qquad i = p+1,\ldots,p+n \tag{4.8}$$

$$\sum_{i=1}^{p+n} x_{ir}^{k} - \sum_{j=1}^{p+n} x_{ij}^{k} = 0 \qquad \begin{matrix} k = 1,\ldots,m \\ r = 1,\ldots,p+n \end{matrix} \tag{4.9}$$

$$\sum_{i=1}^{p} \sum_{j=p+1}^{p+n} x_{ij}^{k} \leq 1 \qquad k = 1,\ldots,m \tag{4.10}$$

$$\sum_{j=1}^{p} \sum_{i=p+1}^{p+n} x_{ij}^{k} \leq 1 \qquad k = 1,\ldots,m \tag{4.11}$$

$$\underline{X} \in S \tag{4.12}$$

$$x_{ij}^{k} \in \{0,1\} \qquad \forall i,j,k \tag{4.13}$$

where S is redefined as one of the three following alternatives

$$S = \{x_{ij} \mid \sum_{i\in Q} \sum_{j\notin Q} x_{ij} \geq 1 \ , \quad \forall Q \subseteq \{1,\ldots,p\}\} \tag{4.12a}$$

$$S = \{x_{ij} \mid \sum_{i\in Q} \sum_{j\in Q} x_{ij} \leq |Q| - 1 \quad \forall Q \subseteq \{p\ 1,\ldots,n\}\} \tag{4.12b}$$

$$S = \{x_{ij} \mid y_j - y_j - nx_{ij} \leq n-1 \ , \ p\ 1 \leq i \neq j \leq n, \ y_i \in \mathbb{R}\} \tag{4.12c}$$

and

$n$ = total number of demand nodes

$p$ = total number of depots

$m$ = total number of vehicles

$C_{ij}$ = cost of traversing arc $(i,j)$

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

$\underline{X}$ = matrix with components $x_{ij} = \sum_{k=1}^{m} x_{ij}^k$ specifying the number of times arc $(i,j)$ is traversed.

The objective function (4.6) states that the total cost is to be minimized. Constraints (4.7) and (4.8) ensure that each demand node is visited by one and only one vehicle. Constraints (4.9) represent the route continuity. They imply that a vehicle entering to a node must exit from that node. The fact that the vehicle availability is not exceeded is made certain by constraints (4.10) and (4.11). Using inequalities, the problem is relaxed in the sense that there is no restriction to employ all the vehicles available. Some of them may be found useless in the optimal solution. Finally, constraints (4.12) can be any of the subtour breaking constraints specified in (4.12a), (4.12b) and (4.12c).

In this model, we assume that vehicle capacities exceed the total demand in the problem and therefore put no restriction on the subtour lengths. As a result, when a demand node is visited, its requirements are satisfied. However, it may be more realistic to include constraints associated with vehicle capacities and total elapsed route time in the model. In this case, these constraints can be included in the model as follows:

$$\sum_{i=1}^{p+n} \sum_{j=p+1}^{p+n} d_j x_{ij}^k \leq P_k \qquad k = 1,\ldots,m \tag{4.14}$$

$$\sum_{i=1}^{p+n} t_i^k \sum_{j=1}^{p+n} x_{ij}^k + \sum_{i=1}^{p+n} \sum_{j=1}^{p+n} t_{ij}^k x_{ij}^k \leq T_k \qquad k = 1,\ldots,m \tag{4.15}$$

Here,

$P_k$ = capacity of vehicle k

$T_k$ = maximum time allowed for a route of vehicle k

$d_j$ = demand at node j

$t_i^t$ = time required for vehicle k to deliver or collect at node i

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

Overall, a careful examination of constraints (4.7)-(4.13) reveals that constraints (4.7) and (4.9) imply constraints (4.8). Similarly, constraints (4.9) and (4.10) imply constraints (4.11). As a consequence of this fact, constraints (4.8) and (4.11) may be excluded from the formulation since they are redundant in solving the problem. Moreover, it should be noted that initial vehicle locations are not being considered by the formulation. The fact that the vehicles must start and

end at the deports where they are initially located is not under control either. But the requirement that at most the given number of vehicles can be used is strongly imposed.

## 4.3 SOLUTION TECHNIQUES FOR THE VEHICLE ROUTING PROBLEMS

Proposed techniques for solving VRPs fall into seven distinct classes as specified by Bodin and Golden [58]:

1. Cluster first - route second
2. route first - cluster second
3. savings/insertion
4. improvement/exchange
5. mathematical programming based
6. interactive optimization
7. exact procedures.

Cluster first-route second procedures group demand nodes first and then design economical routes over each cluster as a second step. Examples of this idea are given by Gillett and Miller [59], Gillett and Johnson [60] and Karp [61] for the standard single depot VRP.

Route first and cluster second procedures construct a large route which includes all the nodes ignoring capacity and range constraints first and then, if infeasible, partition this route into a number of smaller but feasible and economical routes. Examples of route first-cluster second procedures are given by Newton and Thomas [62], Bodin and Berman [63] and Bodin and Kursh [64].

Savings and insertion procedures build a solution in such a way that at each step a comparison is made between the current solution and the alternative solution. The alternative solution is one that yields the largest savings in terms of cost or distance travelled or that inserts a demand entity not existing in the current solution economically. Various savings/insertion procedures for single depot and multiple depot routing problems have been described by Clarke and Wright [65], Golden et.al [66] and Norback and Love [45].

Improvement or exchange procedures such as the heuristics developed by Lin [49], Lin and Kernighan [50] always maintain feasibility and strive towards optimality. At each step, the current feasible solution is altered to yield another feasible solution with a reduced objective function value. The procedure continues until no more improvements are possible. Examples of these procedures can be found in Christofides and Eilon [67] and Bodin and Sexton [68].

Mathematical programming approaches include algorithms that are directly based on a mathematical programming formulation of the underlying model. Examples of this approach can be found in Fisher and Jaikumar [69], Christofides, Mingozzi and Toth [5].

Interactive optimization is a general purpose approach in which an experienced decision maker who has the capability of setting and revising parameters and injecting subjective assessments based on knowledge and intuition is incorporated into the problem-solving process. Adaptations of this approach to the VRP are presented by Krolak, Felts and Marble [70] and Krolak, Felts and Nelson [71].

Exact procedures for solving the VRP include the branch and bound and cutting plane algorithms. However, these procedures have been viable only for small problems. Examples of these procedures which proved to be effective are described by Christofides et.al [5] and Crowder and Padberg [38].

## 4.4 SOLUTION PROCEDURES FOR THE VEHICLE ROUTING PROBLEMS WHICH BUILD UPON THE TRAVELLING SALESMAN PROBLEM AS THE CORE MODEL

It is well known that the TSP is embedded within the most commonly encountered vehicle routing formulations. Two of such problems which are extensions of the TSP, namely the MTSP and the MDVRP were discussed in the previous sections. In this section we will show that although these problems are extensions of the TSP they can be solved as a TSP.

### 4.4.1 The Single Depot Case (MTSP)

As it has been shown by Sweetska and Huckfeldt [8], Bellmore and Hong [6] and the others, it is possible to derive equivalent TSP formulations of the MTSP by the use of a suitable transformation. The transformation is applicable to both symmetric and asymmetric matrices. In summary, it consists of

1. creating m copies of the central depot
2. connecting each of the m copies to the other nodes exactly as the original central depot is connected

3. inserting infinities in the elements of the extended cost matrix which correspond to arcs connecting the copies of the central depot.

For example, consider the MTSP on a complete travelling salesman graph G = (N,E) where N = {1,2,3,4,5}. The associated inter-node cost matrix is given in Table 4.1a. If we let node 5 represent the central depot in which two vehicles are located initially, then the transformation described above yields the cost matrix given in Table 4.1b.

Table 4.1 - Transformation of a cost matrix for the MTSP

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | $\infty$ | $C_{12}$ | $C_{13}$ | $C_{14}$ | $C_{15}$ |
| 2 | $C_{21}$ | $\infty$ | $C_{23}$ | $C_{24}$ | $C_{25}$ |
| 3 | $C_{31}$ | $C_{32}$ | $\infty$ | $C_{34}$ | $C_{35}$ |
| 4 | $C_{41}$ | $C_{42}$ | $C_{43}$ | $\infty$ | $C_{45}$ |
| 5 | $C_{51}$ | $C_{52}$ | $C_{53}$ | $C_{54}$ | $\infty$ |

(a)

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | $\infty$ | $C_{12}$ | $C_{13}$ | $C_{14}$ | $C_{15}$ | $C_{15}$ |
| 2 | $C_{21}$ | $\infty$ | $C_{23}$ | $C_{24}$ | $C_{25}$ | $C_{25}$ |
| 3 | $C_{31}$ | $C_{32}$ | $\infty$ | $C_{34}$ | $C_{35}$ | $C_{35}$ |
| 4 | $C_{41}$ | $C_{42}$ | $C_{43}$ | $\infty$ | $C_{45}$ | $C_{45}$ |
| 5 | $C_{51}$ | $C_{52}$ | $C_{53}$ | $C_{54}$ | $\infty$ | $\infty$ |
| 6 | $C_{51}$ | $C_{52}$ | $C_{53}$ | $C_{54}$ | $\infty$ | $\infty$ |

(b)

Any AP solution using the extended cost matrix and producing m subtours each containing one of the copies created is the optimal MTSP solution. The optimal solution to the MTSP can also be obtained by solving the TSP on the extended cost matrix. As a result, the travelling salesman tour is decomposed into m subtours as required in the MTSP, by coalescing the copies back into a single node. For instance, suppose that the optimal travelling salesman tour obtained by solving the TSP on the

extended matrix is (1-3-5-4-2-6-1). Then, coalescing nodes 5 and 6 back into a single node, namely node 5 in the original problem, yields the subtours (5-1-3-5) and (5-4-2-5) which correspond to individual salesman tours and therefore represent the optimal solution to the MTSP. The back transformation is shown in Figure 4.1.

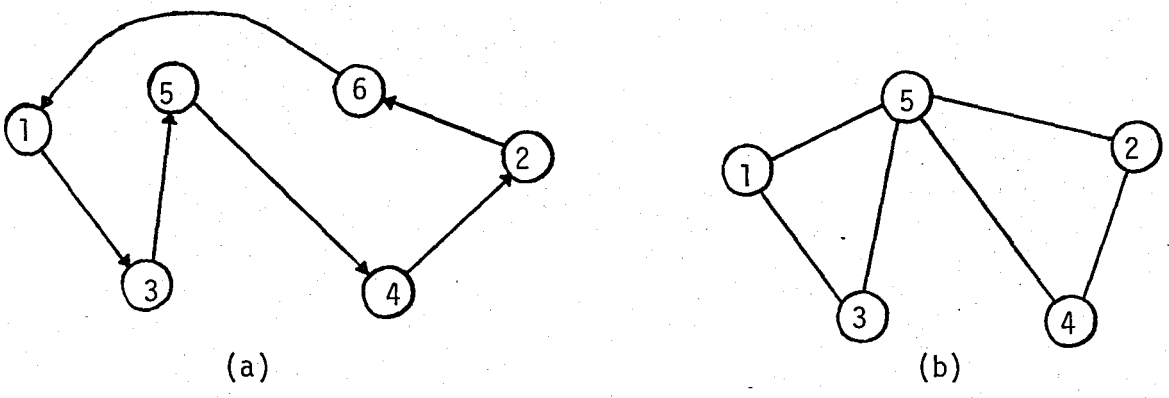



Figure 4.1 - An example of back-transformation for an MTSP.

4.4.2 The Multi-Depot Case (MDVRP)

Similar to the MTSP, the MDVRP can also be converted into an equivalent TSP in a way not principally different from the transformation used for the MTSP. Assuming that we know how the vehicles are located initially, the proposed transformation can be realized by extending both the node set and the arc set together with the associated cost matrix.

4.4.2.1 Transformation of the Node Set

It is clear that the node set of the problem consists of demand nodes and nodes representing the depots. First, the nodes corresponding to the depots are deleted from the node set. Instead, duplicates of the

depots are generated. For each vehicle in a specific depot, two copies of the depot are generated. One of the copies serves as the departure node while the other serves as the arrival node for that particular vehicle. For the sake of simplicity the new nodes are labelled in such a way that nodes labelled with odd numbers represent the departure nodes whereas nodes labelled with even numbers represent the arrival nodes. Considering that there are m vehicles, the number of nodes is increased from (p+n) to (2m+n).

4.4.2.2 Transformation of the Arc Set

First, the arcs connecting the deleted nodes which correspond to the depots are deleted from the arc set. Then, the transformation is executed by

1. connecting the departure node of each vehicle to its arrival node
2. connecting the arrival node of each vehicle to the departure node of another vehicle which has been labelled with a larger number with one exception; The arrival node of the last labelled vehicle is connected to the departure node of the first labelled vehicle
3. connecting each demand node to each arrival node and each departure node to each demand node exactly as the original depots are connected.

As a result, the number of arcs in the arc set of a complete graph is increased by 2m(n+1). After all, there are no arcs entering the departure node except the arc that comes from one of the arrival nodes. Conversely, there are no arcs leaving the arrival nodes except the one incident to one of the departure nodes.

4.4.2.3 Transformation of the Cost Matrix

The cost matrix of the MDVRP is transformed in such a way that the costs of the arcs between the arrival nodes and the demand nodes and between the nodes and the departure nodes are the same as they are for the corresponding depots in the original matrix. The arcs that connect the departure and the arrival nodes are assigned a zero cost. In addition, the costs of the arcs connecting the demand nodes to each other remain the same as they are in the original cost matrix. Finally, infinities are inserted for costs representing nonexistent arcs.

4.4.2.4 An Illustrative Example

Consider an MDVRP defined in a graph G (Figure 4.2a) whose associated cost matrix C is as shown in Table 4.2a. There are eight demand nodes 1,...,8 and two depots 9,10. Suppose that one vehicle is located in each depot. The equivalent travelling salesman graph is shown in Figure 2.4b. After nodes 9 and 10 are deleted four nodes are created. In this case nodes 9 and 11 serve as the departure nodes and nodes 10 and 12 as the arrival nodes. Suppose that the optimal solution to the TSP in the transformed graph is given by (1-2-4-10-11-6-7-8-5-12-9-3-1).

Table 4.2a - The cost matrix

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ∞ | $C_{12}$ | $C_{13}$ | ∞ | $C_{15}$ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 | $C_{21}$ | ∞ | $C_{23}$ | $C_{24}$ | ∞ | ∞ | ∞ | ∞ | $C_{29}$ | ∞ |
| 3 | $C_{31}$ | $C_{32}$ | ∞ | ∞ | $C_{35}$ | ∞ | ∞ | ∞ | $C_{39}$ | ∞ |
| 4 | ∞ | $C_{42}$ | ∞ | ∞ | ∞ | $C_{46}$ | $C_{47}$ | ∞ | $C_{49}$ | $C_{410}$ |
| 5 | $C_{51}$ | ∞ | $C_{53}$ | ∞ | ∞ | ∞ | ∞ | $C_{58}$ | ∞ | $C_{510}$ |
| 6 | ∞ | ∞ | ∞ | $C_{64}$ | ∞ | ∞ | $C_{67}$ | ∞ | ∞ | $C_{610}$ |
| 7 | ∞ | ∞ | ∞ | $C_{74}$ | ∞ | $C_{76}$ | ∞ | $C_{78}$ | ∞ | ∞ |
| 8 | ∞ | ∞ | ∞ | ∞ | $C_{85}$ | ∞ | ∞ | ∞ | ∞ | $C_{810}$ |
| 9 | ∞ | $C_{92}$ | $C_{93}$ | $C_{94}$ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 10 | ∞ | ∞ | ∞ | $C_{104}$ | $C_{105}$ | $C_{106}$ | ∞ | $C_{108}$ | ∞ | ∞ |

Table 4.2b - The transformed cost matrix

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ∞ | $C_{12}$ | $C_{13}$ | ∞ | $C_{15}$ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 | $C_{21}$ | ∞ | $C_{23}$ | $C_{24}$ | ∞ | ∞ | ∞ | ∞ | ∞ | $C_{29}$ | ∞ | ∞ |
| 3 | $C_{31}$ | $C_{32}$ | ∞ | ∞ | $C_{35}$ | ∞ | ∞ | ∞ | ∞ | $C_{39}$ | ∞ | ∞ |
| 4 | ∞ | $C_{42}$ | ∞ | ∞ | ∞ | $C_{46}$ | $C_{47}$ | ∞ | ∞ | $C_{49}$ | ∞ | $C_{410}$ |
| 5 | $C_{51}$ | ∞ | $C_{53}$ | ∞ | ∞ | ∞ | ∞ | $C_{58}$ | ∞ | ∞ | ∞ | $C_{510}$ |
| 6 | ∞ | ∞ | ∞ | $C_{64}$ | ∞ | ∞ | $C_{67}$ | ∞ | ∞ | ∞ | ∞ | $C_{610}$ |
| 7 | ∞ | ∞ | ∞ | $C_{74}$ | ∞ | $C_{76}$ | ∞ | $C_{78}$ | ∞ | ∞ | ∞ | ∞ |
| 8 | ∞ | ∞ | ∞ | ∞ | $C_{85}$ | ∞ | $C_{87}$ | ∞ | ∞ | ∞ | ∞ | $C_{810}$ |
| 9 | ∞ | $C_{92}$ | $C_{93}$ | $C_{94}$ | ∞ | ∞ | ∞ | ∞ | ∞ | 0 | ∞ | ∞ |
| 10 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 0 | ∞ |
| 11 | ∞ | ∞ | ∞ | $C_{104}$ | $C_{105}$ | $C_{106}$ | ∞ | $C_{108}$ | ∞ | ∞ | ∞ | 0 |
| 12 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 0 | ∞ | ∞ | ∞ |

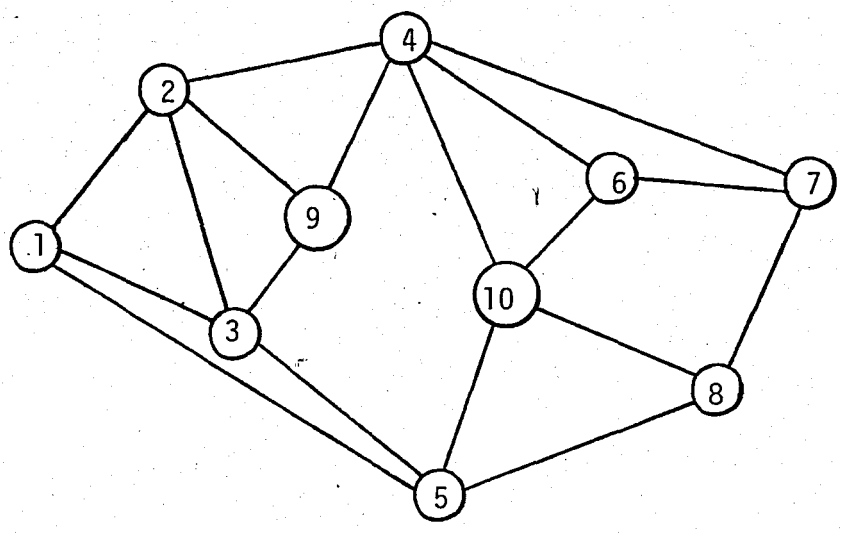



Figure 4.2a - The original graph

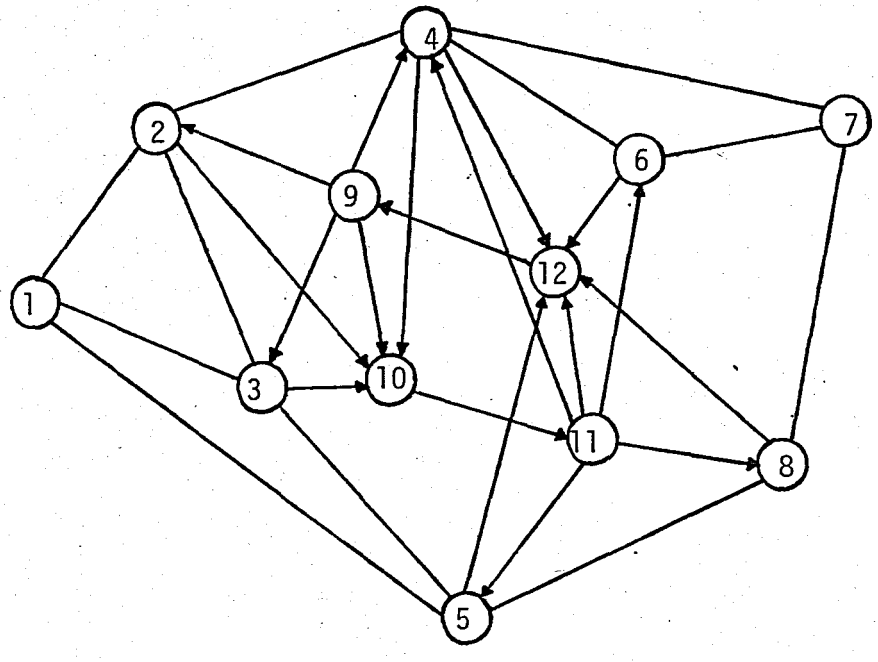



Figure 4.2b - The equivalent travelling salesman graph

Then, coalescing nodes 9, 10 and 11, 12 back into two nodes the travelling salesman tour is decomposed into two subtours corresponding to nodes representing depots 9 and 10 of the original problem. As a result the subtours are: (9-3-1-2-4-9) and (10-6-7-8-5-10) (Figure 4.3a and Figure 4.3b).

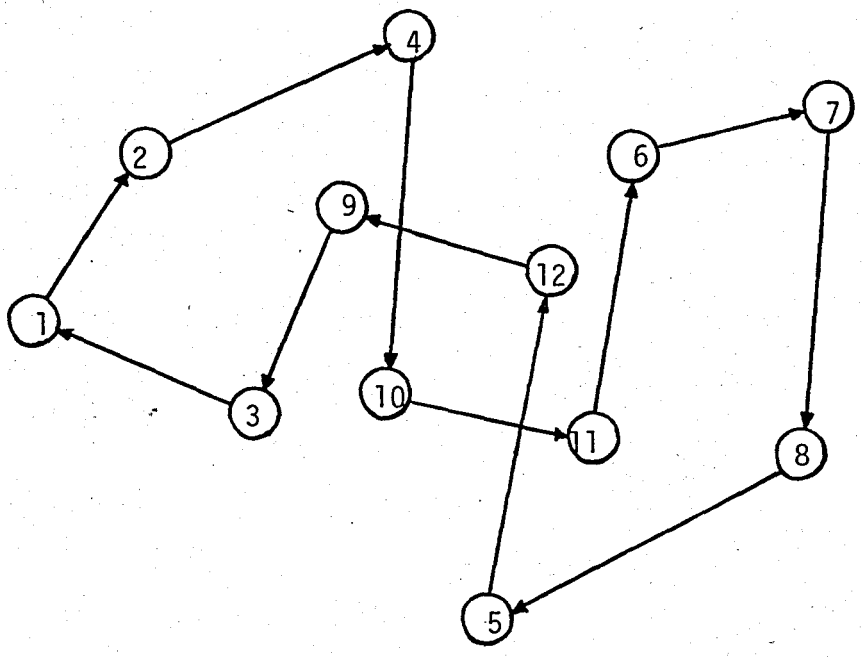



Figure 4.3a - Optimum tour for the travelling salesman graph

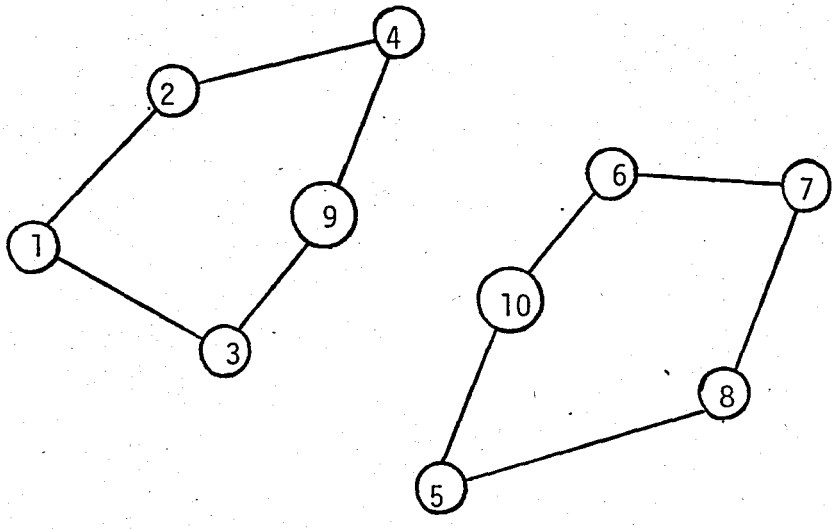



Figure 4.3b - Optimum solution to the MDVRP

4.4.2.5 Equivalence of the Problems

In this section, we will try to prove that the TSP and the MDVRP become two equivalent problems when the transformation described in the previous section is applied. We will show that there is a one to one correspondence between the solutions of both problems. In

addition, the one to one correspondence in ranking the solutions cost-wise will be illustrated.

Since the TSP solution has to cover all the nodes in a graph once and only once, all of the demand nodes will be visited once any only once in the original MDVRP and therefore the demands will be satisfied. Considering the integer programming formulation of the MDVRP, the number of vehicles to be used is bounded from above, but there is no restriction on the number of vehicles that have to be utilized. Actually, the total number of vehicles is an upper bound for the number of subtours in the MDVRP. If, in the TSP solution, the nodes are sequenced in such a way that an arrival node appears immediately after a departure node, then the vehicle corresponding to these nodes is not used in the associated solution of the MDVRP. Hence a one to one correspondence is achieved.

Note that the arcs connecting the arrival nodes to the departure nodes and the arcs connecting the departure nodes to the arrival nodes are assigned zero costs. Besides, the costs corresponding to the arcs which connect the departure nodes to the demand nodes and the demand nodes to the arrival nodes are the same as they are given in the rows and columns of the corresponding depots in the original matrix. As a result, the cost of a travelling salesman tour is exactly the same as the cost of the corresponding solution of MDVRP. Thus, the optimal solution to the TSP in the transformed graph is equivalent to the optimal solution of the MDVRP in the original graph.

# V. APPLICATION OF THE PROPOSED ALGORITHMS TO THE MULTI-DEPOT VEHICLE ROUTING PROBLEM

This section of the thesis is completely devoted to the application of the proposed algorithms to the MDVRP. All of the algorithms are applied to the same problem in order to compare them in equivalent conditions. Actually, the algorithms are expected to perform better for the transformed cost matrix than they do for complete graphs for which examples and computation times are already given in the third chapter. This expectation is due to the special structure of the transformed matrix. In fact, it is certain that all TSP tours will traverse arcs defined between the arrival and departure nodes. Therefore, these arcs need not carried along the search process continuously. This is especially true for the reduction algorithm. But still the other algorithms are also affected by the structure of the transformed matrix.

Consider the MDVRP defined in Figure 5.1. The associated cost matrix, C, is given in Table 5.1. Assume that nodes 8 and 9 represent the depots in which two vehicles are located initially. It is desired to investigate the optimal tours each vehicle should traverse so that

all of the other nodes are visited once only once. We first need to transform the cost matrix in order to apply the proposed algorithms. Accordingly, nodes 8 and 9 are deleted from the problem. Instead two arrival nodes and two departure nodes are created. Each arrival node is connected to a departure node. Similarly, each departure node is connected to an arrival node. Besides all of the demand nodes {1,...,7} are connected to each arrival node and each departure node is connected to the demand nodes exactly as nodes 8 and 9 were connected. The resultant cost matrix C' is given in Table 5.2. Note that nodes 8 and 9 represent the departure and the arrival nodes for the vehicle located in depot 8 in the original problem respectively. Similarly, nodes 10 and 11 represent the departure and the arrival nodes for the vehicle located in depot 9 respectively. Now we are ready for applying the proposed algorithms in order to solve the TSP by using the transformed matrix.

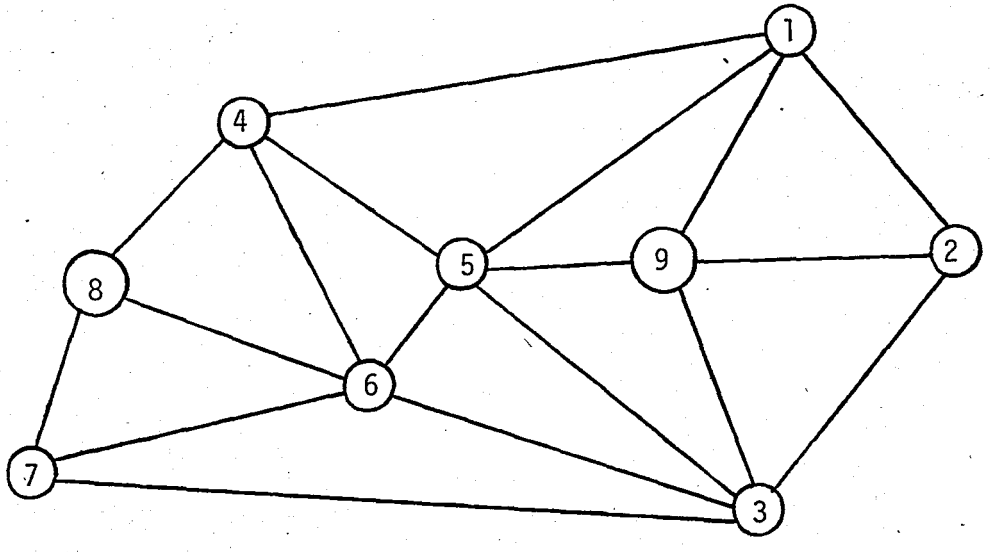



Figure 5.1 - The graph representing the MDVRP

Table 5.1 - The cost matrix corresponding to the MDVRP

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | ∞ | 45 | ∞ | 95 | 70 | ∞ | ∞ | ∞ | 48 |
| 2 | 45 | ∞ | 40 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 3 | ∞ | 40 | ∞ | ∞ | 65 | 70 | 125 | ∞ | 40 |
| 4 | 95 | ∞ | ∞ | ∞ | 43 | 48 | ∞ | 33 | ∞ |
| 5 | 70 | ∞ | 65 | 43 | ∞ | 25 | ∞ | ∞ | 28 |
| 6 | ∞ | ∞ | 70 | 48 | 25 | ∞ | 60 | 47 | ∞ |
| 7 | ∞ | ∞ | 125 | ∞ | ∞ | 60 | ∞ | 25 | ∞ |
| 8 | ∞ | ∞ | ∞ | 33 | ∞ | 47 | 25 | ∞ | ∞ |
| 9 | 48 | ∞ | 40 | ∞ | 28 | ∞ | ∞ | ∞ | ∞ |

Table 5.2 - The transformed cost matrix

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ∞ | 45 | ∞ | 95 | 70 | ∞ | ∞ | ∞ | ∞ | ∞ | 48 |
| 2 | 45 | ∞ | 40 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 3 | ∞ | 40 | ∞ | ∞ | 65 | 70 | 125 | ∞ | ∞ | ∞ | 40 |
| 4 | 95 | ∞ | ∞ | ∞ | 43 | 48 | ∞ | ∞ | 33 | ∞ | ∞ |
| 5 | 70 | ∞ | 65 | 43 | ∞ | 25 | ∞ | ∞ | ∞ | ∞ | 28 |
| 6 | ∞ | ∞ | 70 | 48 | 25 | ∞ | 60 | ∞ | 47 | ∞ | ∞ |
| 7 | ∞ | ∞ | 125 | ∞ | ∞ | 60 | ∞ | ∞ | 25 | ∞ | ∞ |
| 8 | ∞ | ∞ | ∞ | 33 | ∞ | 47 | 25 | ∞ | 0 | ∞ | ∞ |
| 9 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 0 | ∞ |
| 10 | 48 | ∞ | 40 | ∞ | 28 | ∞ | ∞ | ∞ | ∞ | ∞ | 0 |
| 11 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 0 | ∞ | ∞ | ∞ |

## 5.1 APPLICATION OF ALGORITHM I

As a result of reducing the cost matrix C' the corresponding subgraph G' is shown in Figure 5.2a. Note that G' is not connected. The algorithm proceeds as follows:

Choose node 1. $R(1) = \{1,2,3,7,8,9,10,11\} \neq N$

$$\pi_1 = \underset{\substack{i\in R(1)\\ j\in N-R(1)}}{min} \{C'_{ij}\} = C'_{84} = 15$$

Including arc (8,4) in G' (Figure 5.2b) we see that G' is still disconnected.

$R(1) = \{1,2,3,4,7,8,9,10,11\} \neq N$

$$\pi_1 = \underset{\substack{i\in R(1)\\ j\in N-R(1)}}{min} \{C'_{ij}\} = C'_{15} = C'_{35} = C'_{45} = 10$$

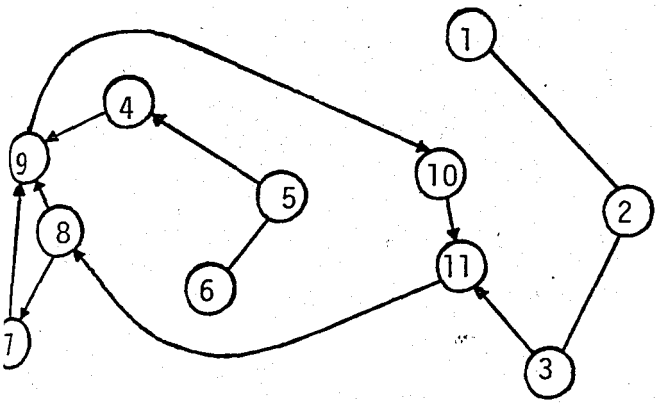
Once arcs (1,5), (3,5), (4,5) are included in G' we obtain $R(1) = N$. The resultant subgraph is shown in Figure 5.2c.

Choose node 2. $R(2) = N$

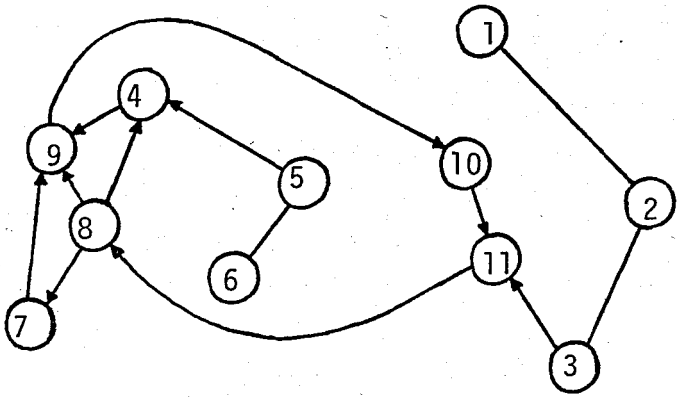
Choose node 3. $R(3) = N$

Choose node 4. $R(4) = \{4,5,6,7,8,9,10,11\} \neq N$

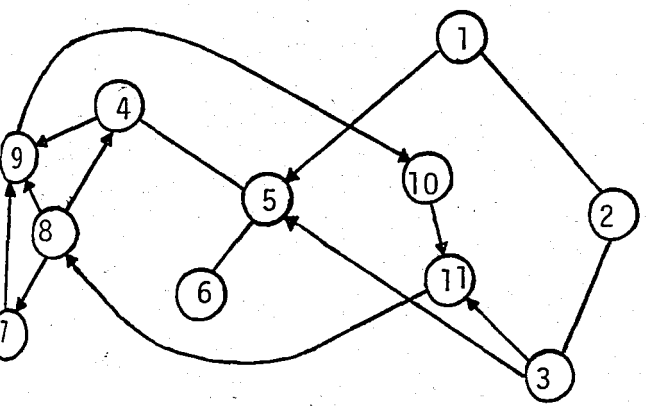
$$\pi_4 = \underset{\substack{i\in R(4)\\ j\in N-N(4)}}{min} \{C'_{ij}\} = C'_{51} = C'_{53} = C'_{103} = 40$$



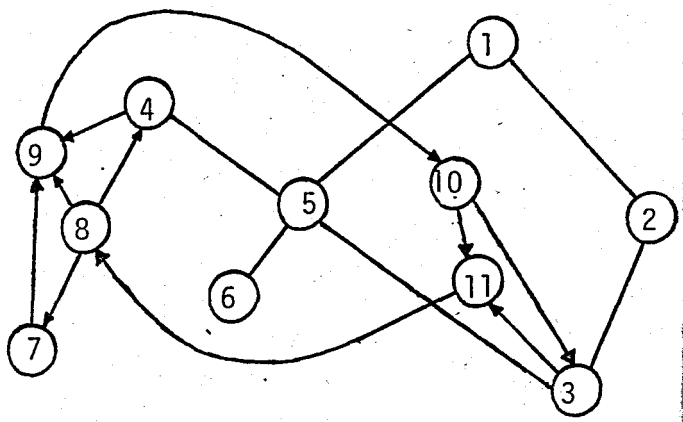

(a)




(b)




(c)




(d)

Figure 5.2 - Stages in constructing the subgraph* G'

After including arcs (5,1), (5,3) (10,3) in G', the set of nodes reachible from node 4 becomes equivalent to N, i.e. R(4) = N. The resultant subgraph G' is shown in Figure 5.2d.

*Lines without arrows denote arcs in two directions.

Choose node 5. R(5) = N.

Choose node 6. R(6) = N.

Choose node 7. R(7) = N.

Choose node 8. R(8) = N.

Choose node 9. R(9) = N.

Choose node 10. R(10) = N.

Choose node 11. R(11) = N.

Since all nodes have been tried and strong connectedness is achieved we proceed by checking the unilateral connectedness in any subgraph $G'_k$ obtained by removing a node k and the associated arcs from G'.

Remove node 1. $G'_1$ is unilaterally connected (Fig. 5.3a).

Remove node 2. $G'_2$ is unilaterally connected (Fig. 5.3b).

Remove node 3. $G'_3$ is unilaterally connected (Fig. 5.3c).

Remove node 4. $G'_4$ is unilaterally connected (Fig. 5.3d).

Remove node 5. $G'_5$ is not unilaterally connected (Fig. 5.3e).

There is no path either from node 1 to node 6 or from node 6 to node 1.

$R(1) = \{1,2,3,4,7,8,9,10,11\}$ $\quad R(6) = \{6\}$

$N-\{5\}-R(1) = \{6\}$ $\quad N-\{5\}-R(6) = \{1,2,3,4,7,8,9,10,11\}$

$$\mu_{16} = \min\{\min_{\substack{i\in R(1)\\ j\in N-\{5\}-R(1)}} \{C'_{ij}\},\ \min_{\substack{i\in R(6)\\ j\in N-\{5\}-R(6)}} \{C'_{ij}\}\} = C'_{36}=C'_{46}=C'_{63}=C'_{64}= 5$$

The updated $G'_5$ is shown in Fig. 5.3f. Note that $G'_5$ becomes unilaterally connected.

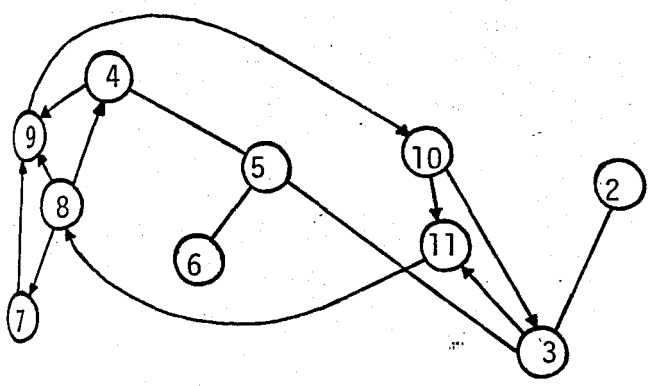



Figure 5.3a - Subgraph $G'_1$

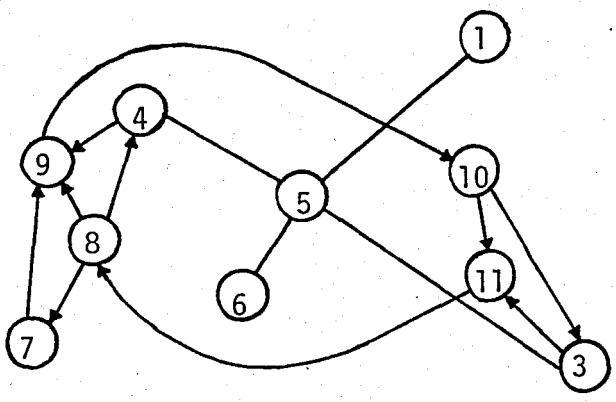



Figure 5.3b - Subgraph $G'_2$

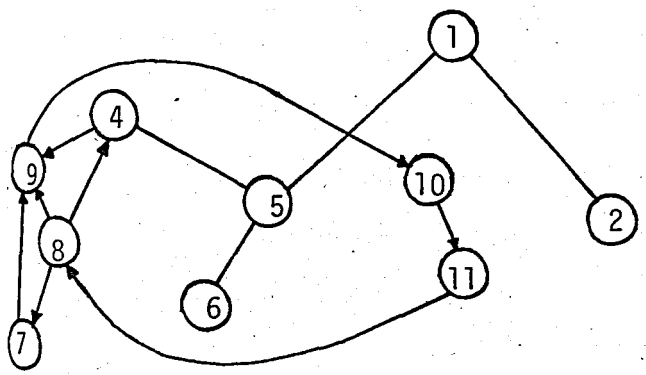



Figure 5.3c - Subgraph $G'_3$

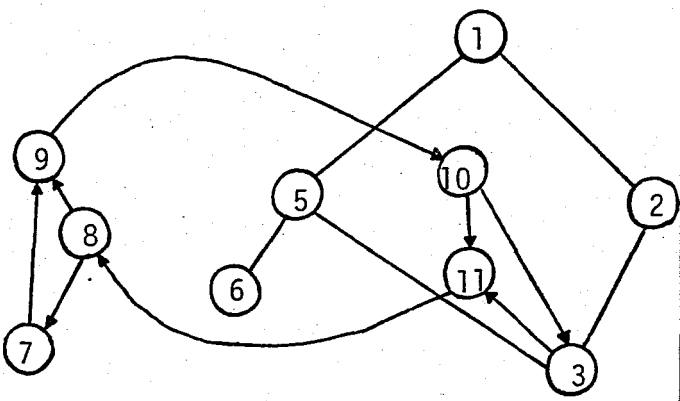



Figure 5.3d - Subgraph $G'_4$

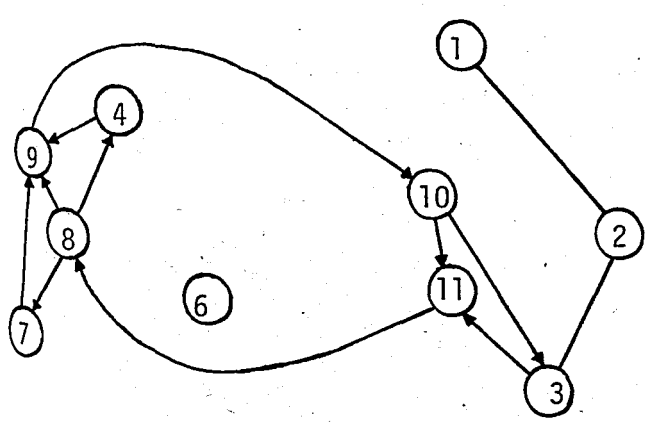



Figure 5.3e - Subgraph $G'_5$

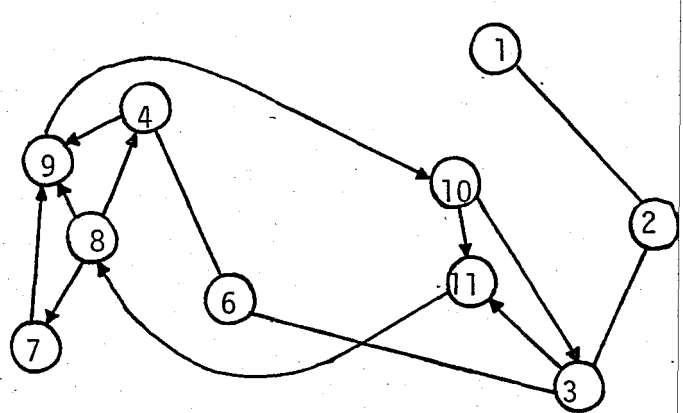



Figure 5.3f - Subgraph $G'_5$

Remove node 6. $G_6'$ is unilaterally connected (Fig. 5.3g).

Remove node 7. $G_7'$ is unilaterally connected (Fig. 5.3h).

Remove node 8. $G_8'$ is unilaterally connected (Fig. 5.3i).

Remove node 9. $G_9'$ is unilaterally connected (Fig. 5.3j).

Remove node 10. $G_{10}'$ is unilaterally connected (Fig. 5.3k).

Remove node 11. $G_{11}'$ is unilaterally connected (Fig. 5.3 ).

Unfortunately at this stage G' does not possess any Hamiltonian circuit (Figure 5.4); therefore we proceed with applying Little's algorithm partially to the resultant matrix which is given in Table 5.3. We will not go over the steps of the algorithm but instead give the solution found which is (1-4-6-5-11-8-7-9-10-3-2-1) with a total cost of 371. Coalescing nodes 8 and 9 and nodes 10 and 11 back into depots 8 and 9 in the original problem respectively, we obtain two subtours with the same total cost. The tours are

Tour 1 = (8-7-8)

Tour 2 = (9-3-2-1-4-6-5-9)

Total cost = 371

The tours are shown in Figure 5.7a.

## 5.2 APPLICATION OF ALGORITHM II

Starting with the transformed matrix we apply the second algorithm as follows: According to the first step of the algorithm we solve the AP. The resultant matrix is given in Table 5.4. The associated subtours and the corresponding penalties are shown in Figure 5.5a. Starting from the minimum of the penalties that would

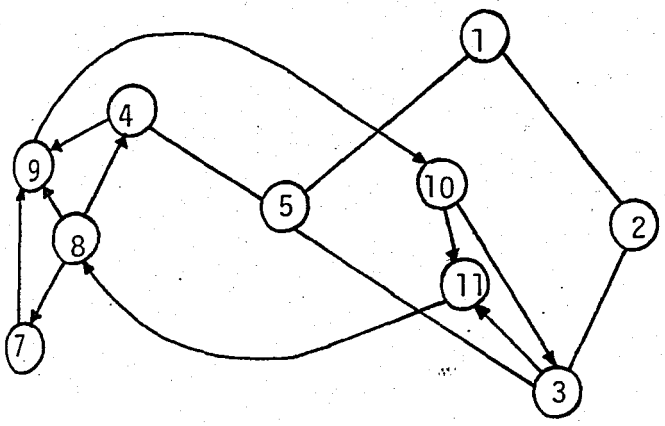



Figure 5.3g - Subgraph $G'_6$

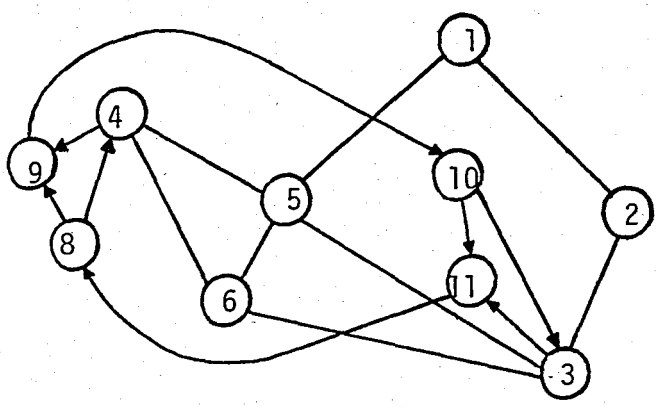



Figure 5.3h - Subgraph $G'_7$

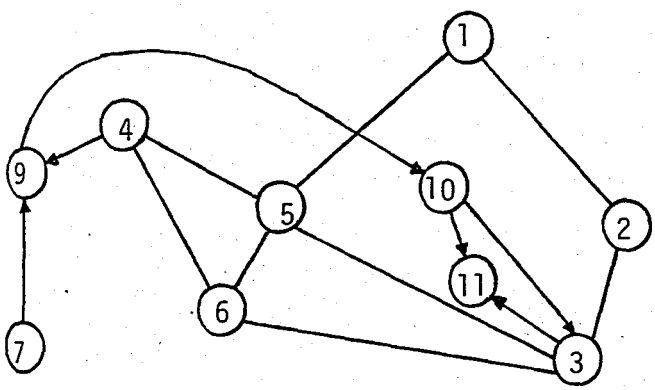



Figure 5.3i - Subgraph $G'_8$

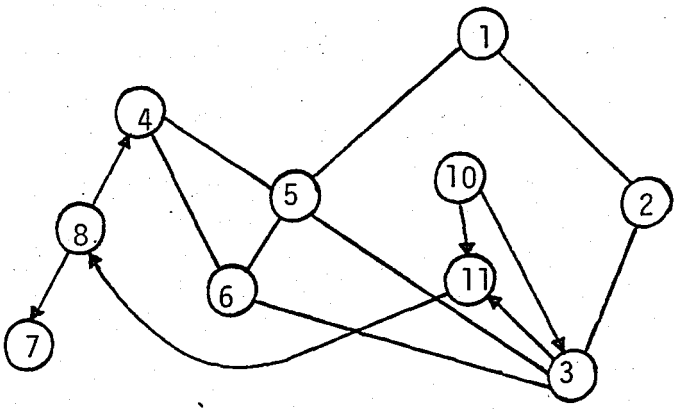



Figure 5.3j - Subgraph $G'_9$

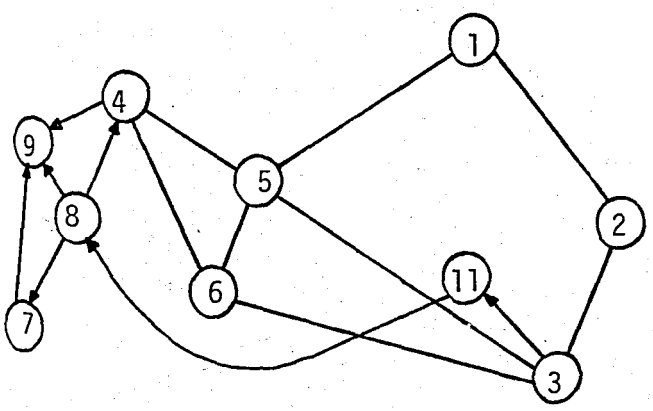



Figure 5.3k - Subgraph $G'_{10}$

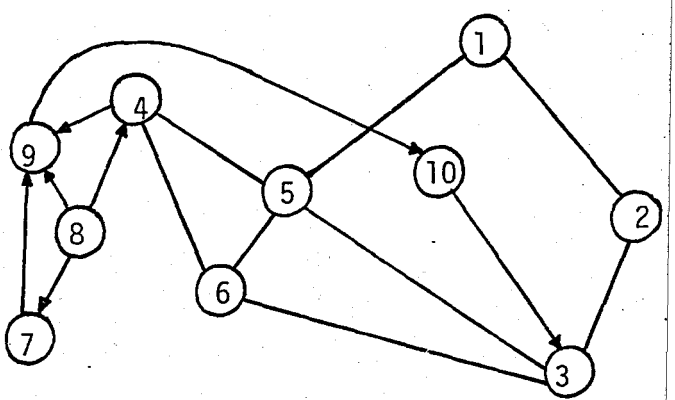



Figure 5.3l - Subgraph $G'_{11}$

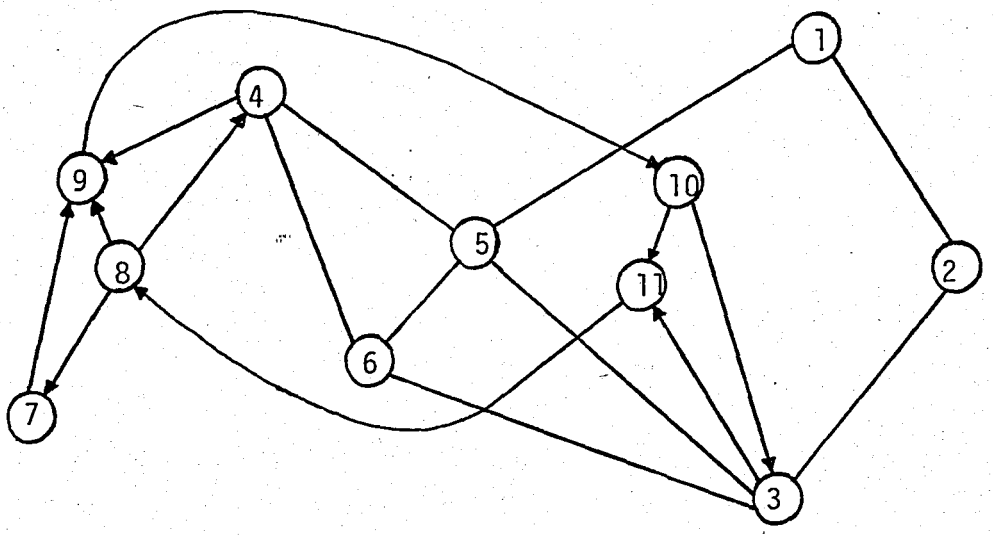


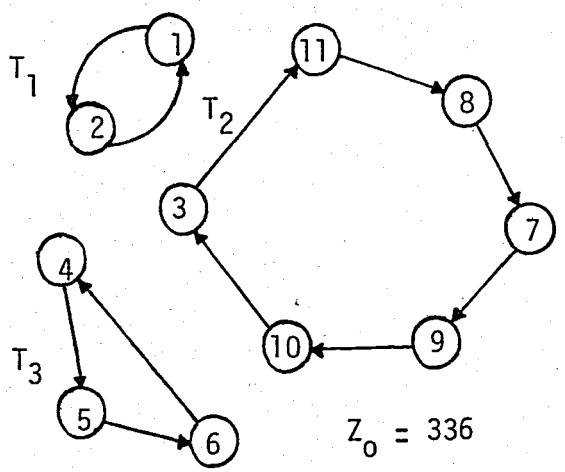

Figure 5.4 - The resultant subgraph G'

Table 5.3 - The resultant reduced transformed matrix

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ∞ | 0 | ∞ | 17 | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | 3 |
| 2 | 0 | ∞ | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 3 | ∞ | 0 | ∞ | ∞ | 0 | 0 | 60 | ∞ | ∞ | ∞ | 0 |
| 4 | 17 | ∞ | ∞ | ∞ | 0 | 0 | ∞ | ∞ | 0 | ∞ | ∞ |
| 5 | 0 | ∞ | 0 | 0 | ∞ | 0 | ∞ | ∞ | ∞ | ∞ | 3 |
| 6 | ∞ | ∞ | 0 | 0 | 0 | ∞ | 5 | ∞ | 17 | ∞ | ∞ |
| 7 | ∞ | ∞ | 60 | ∞ | ∞ | 5 | ∞ | ∞ | 0 | ∞ | ∞ |
| 8 | ∞ | ∞ | ∞ | 0 | ∞ | 17 | 0 | ∞ | 0 | ∞ | ∞ |
| 9 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 0 | ∞ |
| 10 | 3 | ∞ | 0 | ∞ | 3 | ∞ | ∞ | ∞ | ∞ | ∞ | 0 |
| 11 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 0 | ∞ | ∞ | ∞ |

Table 5.4 - The transformed cost matrix after the AP is solved in the first step of algorithm II

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ∞ | 0 | ∞ | 2 | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | 3 |
| 2 | 0 | ∞ | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 3 | ∞ | 0 | ∞ | ∞ | 0 | 0 | 25 | ∞ | ∞ | ∞ | 0 |
| 4 | 32 | ∞ | ∞ | ∞ | 0 | 0 | ∞ | ∞ | 0 | ∞ | ∞ |
| 5 | 30 | ∞ | 30 | 0 | ∞ | 0 | ∞ | ∞ | ∞ | ∞ | 33 |
| 6 | ∞ | ∞ | 30 | 0 | 0 | ∞ | 0 | ∞ | 32 | ∞ | ∞ |
| 7 | ∞ | ∞ | 75 | ∞ | ∞ | 20 | ∞ | ∞ | 0 | ∞ | ∞ |
| 8 | ∞ | ∞ | ∞ | 20 | ∞ | 52 | 0 | ∞ | 20 | ∞ | ∞ |
| 9 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 0 | ∞ |
| 10 | 3 | ∞ | 0 | ∞ | 3 | ∞ | ∞ | ∞ | ∞ | ∞ | 0 |
| 11 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 0 | ∞ | ∞ | ∞ |




$\bar{p}_{12} = 0$ $\bar{p}_{79} = 20$

$\bar{p}_{21} = 3$ $\bar{p}_{87} = 20$

$\bar{p}_{311} = 0$ $\bar{p}_{910} = \infty$

$\bar{p}_{45} = 0$ $\bar{p}_{103} = 3$

$\bar{p}_{56} = 30$ $\bar{p}_{118} = \infty$

$\bar{p}_{64} = 0$

Figure 5.5a - Subtours and penalties corresponding to the AP solution

be incurred if the assignments are not to be made, we solve the corresponding APs until the best solution found $Z_1^*$ is less than the next penalty to be considered. The following solutions are obtained in each case:

1. $C'_{1j} = \infty \quad \forall j \in T_1$

   {(1,11),(2,1),(3,2),(4,5),(5,6),(6,4),(7,9),(8,7),(9,10),
   (10,3),(11,8)}

   $Z_1 = 3 \quad Z_1^* = 3$

2. $C'_{2j} = \infty \quad \forall j \in T_2$

   {(1,2),(2,3),(3,6),(4,5),(5,1),(6,4),(7,9),(8,7),(9,10),
   (10,11),(11,8)}

   $Z_1 = 30 \quad Z_1^* = 3$

3. $C'_{4j} = \infty \quad \forall j \in T_3$

   {(1,2),(2,1),(3,11),(4,9),(5,4),(6,5),(7,6),(8,7),(9,10),
   (10,3),(11,8)}

   $Z_1 = 20 \quad Z_1^* = 3$

4. $C'_{6j} = \infty \quad \forall j \in T_3$

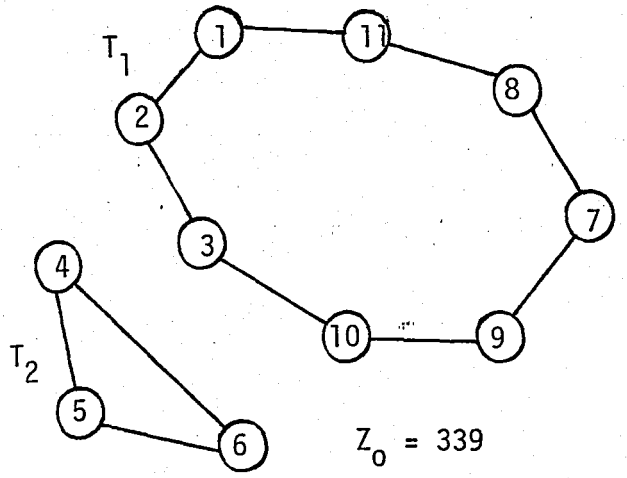
   {(1,2),(2,1),(3,11),(4,5),(5,6),(6,7),(7,9),(8,4),(9,10),
   (10,3),(11,8)}

   $Z_1 = 20 \quad Z_1^* = 3$

Since $Z_1^*$ equals to $\bar{p}_{31} = 3$ which is the next penalty to be considered, we do not need to solve any other AP at this point. Instead, we update the cost matrix, i.e. take the one which corresponds to $Z_1 = 3$ and calculate the new penalties. The associated cost matrix is given in Table 5.5. Defining the subtours, the corresponding penalties are as shown in Figure 5.5b. Note that $Z_o = 339$ becomes the new objective function value of the original problem. The following AP solutions are obtained by solving an AP for each penalty in rank:

Table 5.5 - The cost matrix that corresponds to solution (1)

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ∞ | -3 | ∞ | 2 | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | 0 |
| 2 | 0 | ∞ | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 3 | ∞ | 0 | ∞ | ∞ | 3 | 3 | 28 | ∞ | ∞ | ∞ | 0 |
| 4 | 29 | ∞ | ∞ | ∞ | 0 | 0 | ∞ | ∞ | 0 | ∞ | ∞ |
| 5 | 27 | ∞ | 27 | 0 | ∞ | 0 | ∞ | ∞ | ∞ | ∞ | 33 |
| 6 | ∞ | ∞ | 27 | 0 | 0 | ∞ | 0 | ∞ | 32 | ∞ | ∞ |
| 7 | ∞ | ∞ | 75 | ∞ | ∞ | 20 | ∞ | ∞ | 0 | ∞ | ∞ |
| 8 | ∞ | ∞ | ∞ | 23 | ∞ | 35 | 0 | ∞ | 23 | ∞ | ∞ |
| 9 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 0 | ∞ |
| 10 | 3 | ∞ | 0 | ∞ | 6 | ∞ | ∞ | ∞ | ∞ | ∞ | 0 |
| 11 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 0 | ∞ | ∞ | ∞ |




$Z_o = 339$

$\bar{p}_{111} = 0$ $\quad \bar{p}_{79} = 20$

$\bar{p}_{21} = \infty$ $\quad \bar{p}_{87} = 23$

$\bar{p}_{32} = 0$ $\quad \bar{p}_{910} = \infty$

$\bar{p}_{45} = 0$ $\quad \bar{p}_{103} = 6$

$\bar{p}_{56} = 27$ $\quad \bar{p}_{118} = \infty$

$\bar{p}_{64} = 0$

Figure 5.5b - Subtours and penalties corresponding to the AP solution (1)

1. $C'_{1j} = \infty \quad \forall j \in T_1$

{(1,5),(2,1),(3,2),(4,6),(5,4),(6,3),(7,9),(8,7),(9,10),(10,11),(11,8)}

$Z_1 = 27 \quad Z_1^* = 27$

2. $C'_{3j} = \infty \quad \forall j \in T_1$

{(1,2),(2,1),(3,6),(4,5),(5,4),(6,3),(7,9),(8,7),(9,10),(10,11),(11,8)}

$Z_1 = 27 \quad Z_1^* = 27$

3. $C'_{4j} = \infty \quad \forall j \in T_2$

{(1,11),(2,1),(3,2),(4,9),(5,4),(6,5),(7,6),(8,7),(9,10),(10,3),(11,8)}

$Z_1 = 20 \quad Z_1^* = 20$

4. $C'_{6j} = \infty \quad \forall j \in T_2$

{(1,11),(2,1),(3,2),(4,5),(5,6),(6,7),(7,9),(8,4),(9,10),(10,3),(11,8)}

$Z_1 = 20 \quad Z_1^* = 20$

5. $C'_{10j} = \infty \quad \forall j \in T_1$

$\{(1,11),(2,1),(3,2),(4,6),(5,3),(6,4),(7,9),(8,7),(9,10),(10,5),(11,8)\}$

$Z_1 = 33 \qquad Z_1^* = 20$

At this point, we do not need to proceed with solving any other AP since $Z_1^*$ equals to the next penalty to be considered. On the other hand, solutions (3) and (4), both of which have the objective function value $Z_1 = Z_1^* = 20$, are travelling salesman tours. Therefore, the best achievable solution is obtained with $Z_0 = 359$. It should be noted that the fact that there are two solutions with the same objective function value in this case is a consequence of the symmetric nature of the original cost matrix. In other words, both of the solutions correspond to the same subtours in the original problem. Choosing solution (3) the best achievable tour is expressed as (1-11-8-7-6-5-4-9-10-3-2-1). Coalescing the vehicle departure and arrival nodes back into single depots the following tours are obtained (Figure 5.7b).

Tour 1 = (8-7-6-5-4-8)

Tour 2 = (9-3-2-1-9)

Total cost = 359

## 5.3 APPLICATION OF ALGORITHM III

As required by the third algorithm, all of the elements $C'_{ij}$ are subtracted from a large number L which is chosen to be 250 in this case. The resultant matrix is given in Table 5.6.

Table 5.6 - The cost matrix after subtracting each element from a large number L = 250

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ∞ | 205 | ∞ | 155 | 180 | ∞ | ∞ | ∞ | ∞ | ∞ | 202 |
| 2 | 205 | ∞ | 210 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 3 | ∞ | 210 | ∞ | ∞ | 185 | 180 | 125 | ∞ | ∞ | ∞ | 210 |
| 4 | 155 | ∞ | ∞ | ∞ | 207 | 202 | ∞ | ∞ | 217 | ∞ | ∞ |
| 5 | 180 | ∞ | 185 | 207 | ∞ | 225 | ∞ | ∞ | ∞ | ∞ | 222 |
| 6 | ∞ | ∞ | 180 | 202 | 225 | ∞ | 190 | ∞ | 203 | ∞ | ∞ |
| 7 | ∞ | ∞ | 125 | ∞ | ∞ | 190 | ∞ | ∞ | 225 | ∞ | ∞ |
| 8 | ∞ | ∞ | ∞ | 217 | ∞ | 203 | 225 | ∞ | 250 | ∞ | ∞ |
| 9 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 250 | ∞ |
| 10 | 202 | ∞ | 210 | ∞ | 222 | ∞ | ∞ | ∞ | ∞ | ∞ | 250 |
| 11 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 250 | ∞ | ∞ | ∞ |

As a result, the algorithm proceeds as follows:

Step (1) $s = 1$, $S = \{2,4,5,11\}$, $k = 1$

| | | |
|---|---|---|
| $\ell^1(2) = 205$ | $\theta^1(2) = 1$ | $p^1(2) = \{1,2\}$ |
| $\ell^1(3) = 0$ | $\theta^1(3) = 1$ | $p^1(3) = \phi$ |
| $\ell^1(4) = 155$ | $\theta^1(4) = 1$ | $p^1(4) = \{1,4\}$ |
| $\ell^1(5) = 180$ | $\theta^1(5) = 1$ | $p^1(5) = \{1,5\}$ |
| $\ell^1(6) = 0$ | $\theta^1(6) = 1$ | $p^1(6) = \phi$ |
| $\ell^1(7) = 0$ | $\theta^1(7) = 1$ | $p^1(7) = \phi$ |
| $\ell^1(8) = 0$ | $\theta^1(8) = 1$ | $p^1(8) = \phi$ |
| $\ell^1(9) = 0$ | $\theta^1(9) =$ | $p^1(9) = \phi$ |
| $\ell^1(10) = 0$ | $\theta^1(10) = 1$ | $p^1(10) = \phi$ |
| $\ell^1(11) = 202$ | $\theta^1(11) = 1$ | $p^1(11) = \{1,11\}$ |

Step 2 R(S) = {1,3,4,5,6,8,9,11}

| | | |
|---|---|---|
| $\ell^2(2) = 205$ | $\theta^2(2) = 1$ | $p^2(2) = \{1,2\}$ |
| $\ell^2(3) = 415$ | $\theta^2(3) = 2$ | $p^2(3) = \{1,2,3\}$ |
| $\ell^2(4) = 387$ | $\theta^2(4) = 5$ | $p^2(4) = \{1,4,5\}$ |
| $\ell^2(5) = 362$ | $\theta^2(5) = 4$ | $p^2(5) = \{1,4,5\}$ |
| $\ell^2(6) = 405$ | $\theta^2(6) = 5$ | $p^2(6) = \{1,5,6\}$ |
| $\ell^2(7) = 0$ | $\theta^2(7) = 1$ | $p^2(7) = \phi$ |
| $\ell^2(8) = 452$ | $\theta^2(8) = 11$ | $p^2(8) = \{1,8,11\}$ |
| $\ell^2(9) = 372$ | $\theta^2(9) = 4$ | $p^2(9) = \{1,4,9\}$ |
| $\ell^2(10) = 0$ | $\theta^2(10) = 1$ | $p^2(10) = \phi$ |
| $\ell^2(11) = 402$ | $\theta^2(11) = 5$ | $p^2(11) = \{1,5,11\}$ |

Step (3) $k < 9$, continue

Step (4) S = {3,4,5,6,8,9,11}

Step (5) k = 2

Step (2) R(S) = {1,2,3,4,5,6,7,8,9,10,11}

| | | |
|---|---|---|
| $\ell^3(2) = 205$ | $\theta^3(2) = 2$ | $p^3(2) = \{1,2\}$ |
| $\ell^3(3) = 585$ | $\theta^3(3) = 6$ | $p^3(3) = \{1,3,5,6\}$ |
| $\ell^3(4) = 669$ | $\theta^3(4) = 8$ | $p^3(4) = \{1,4,8,11\}$ |
| $\ell^3(5) = 600$ | $\theta^3(5) = 3$ | $p^3(5) = \{1,2,3,5\}$ |
| $\ell^3(6) = 655$ | $\theta^3(6) = 8$ | $p^3(6) = \{1,6,8,11\}$ |
| $\ell^3(7) = 677$ | $\theta^3(7) = 8$ | $p^3(7) = \{1,7,8,11\}$ |
| $\ell^3(8) = 652$ | $\theta^3(8) = 11$ | $p^3(8) = \{1,5,8,11\}$ |
| $\ell^3(9) = 702$ | $\theta^3(9) = 8$ | $p^3(9) = \{1,8,9,11\}$ |
| $\ell^3(10) = 622$ | $\theta^3(10) = 9$ | $p^3(10) = \{1,4,9,10\}$ |
| $\ell^3(11) = 625$ | $\theta^3(11) = 3$ | $p^3(11) = \{1,2,3,11\}$ |

Step (3) $k < 9$, continue

Step (4) $S = \{3,4,5,6,7,8,9,10,11\}$

Step (5) $k = 3$

Step (2) $R(S) = \{1,2,3,4,5,6,7,8,9,10,11\}$

| | | |
|---|---|---|
| $\ell^4(2) = 795$ | $\theta^4(2) = 3$ | $p^4(2) = \{1,2,3,5,6\}$ |
| $\ell^4(3) = 835$ | $\theta^4(3) = 6$ | $p^4(3) = \{1,3,6,8,11\}$ |
| $\ell^4(4) = 869$ | $\theta^4(4) = 8$ | $p^4(4) = \{1,4,5,8,11\}$ |
| $\ell^4(5) = 880$ | $\theta^4(5) = 6$ | $p^4(5) = \{1,5,6,8,11\}$ |
| $\ell^4(6) = 871$ | $\theta^4(6) = 4$ | $p^4(6) = \{1,4,6,8,11\}$ |
| $\ell^4(7) = 877$ | $\theta^4(7) = 8$ | $p^4(7) = \{1,5,7,8,11\}$ |
| $\ell^4(8) = 875$ | $\theta^4(8) = 11$ | $p^4(8) = \{1,2,3,8,11\}$ |
| $\ell^4(9) = 902$ | $\theta^4(9) = 7$ | $p^4(9) = \{1,7,8,9,11\}$ |
| $\ell^4(10) = 952$ | $\theta^4(10) = 9$ | $p^4(10) = \{1,8,9,10,1\}\}$ |
| $\ell^4(11) = 872$ | $\theta^4(11) = 10$ | $p^4(11) = \{1,4,9,10,11\}$ |

Step (3) $k < 9$, continue

Step (4) $S = \{2,3,4,5,6,7,8,9,10,11\}$

Step (5) $k = 4$

Step (2) $R(S) = \{1,2,3,4,5,6,7,8,9,10,11\}$

$\ell^5(2) = 1045$ $\theta^5(2) = 3$ $p^5(2) = \{1,2,3,6,8,11\}$

$\ell^5(3) = 1162$ $\theta^5(3) = 10$ $p^5(3) = \{1,3,8,9,10,11\}$

$\ell^5(4) = 1092$ $\theta^5(4) = 8$ $p^5(4) = \{1,2,3,4,8,11\}$

$\ell^5(5) = 1174$ $\theta^5(5) = 10$ $p^5(5) = \{1,5,8,9,10,11\}$

$\ell^5(6) = 1078$ $\theta^5(6) = 8$ $p^5(6) = \{1,2,3,6,8,11\}$

$\ell^5(7) = 1100$ $\theta^5(7) = 8$ $p^5(7) = \{1,2,3,7,8,11\}$

$\ell^5(8) = 1122$ $\theta^5(8) = 11$ $p^5(8) = \{1,4,8,9,10,11\}$

$\ell^5(9) = 1125$ $\theta^5(9) = 8$ $p^5(9) = \{1,2,3,8,9,11\}$

$\ell^5(10) = 1152$ $\theta^5(10) = 9$ $p^5(10) = \{1,7,8,9,10,11\}$

$\ell^5(11) = 872$ $\theta^5(11) = 11$ $p^5(11) = \{1,4,9,10,11\}$

Step (3) $k < 9$, continue

Step (4) $S = \{2,3,4,5,6,7,8,9,10\}$

Step (5) $k = 5$

Step (2) $R(S) = \{1,2,3,4,5,6,7,9,10,11\}$

$\ell^6(2) = 1372$ $\theta^6(2) = 3$ $p^6(2) = \{1,2,3,8,9,10,11\}$

$\ell^6(3) = 1362$ $\theta^6(3) = 10$ $p^6(3) = \{1,3,7,8,9,10,11\}$

$\ell^6(4) = 1381$ $\theta^6(4) = 5$ $p^6(4) = \{1,4,5,8,9,10,11\}$

$\ell^6(5) = 1374$ $\theta^6(5) = 10$ $p^6(5) = \{1,5,7,8,9,10,11\}$

$\ell^6(6) = 1399$ $\theta^6(6) = 5$ $p^6(6) = \{1,5,6,8,9,10,11\}$

$\ell^6(7) = 1347$ $\theta^6(7) = 8$ $p^6(7) = \{1,4,7,8,9,10,11\}$

$\ell^6(8) = 1122$ $\theta^6(8) = 8$ $p^6(8) = \{1,4,8,9,10,11\}$

$\ell^6(9) = 1325$ $\theta^6(9) = 7$ $p^6(9) = \{1,2,3,7,8,9,11\}$

$\ell^6(10) = 1375$ $\theta^6(10) = 9$ $p^6(10) = \{1,2,3,8,9,10,11\}$

$\ell^6(11) = 872$ $\theta^6(11) = 11$ $p^6(11) = \{1,4,9,10,11\}$

Step (3) $k < 9$, continue

Step (4) $S = \{2,3,4,5,6,7,9,10\}$

Step (5) $k = 6$

Step (2) $R(S) = \{1,2,3,4,5,6,7,9,10,11\}$

| | | |
|---|---|---|
| $\ell^7(2) = 1572$ | $\theta^7(2) = 3$ | $p^7(2) = \{1,2,3,7,8,9,10,11\}$ |
| $\ell^7(3) = 1579$ | $\theta^7(3) = 6$ | $p^7(3) = \{1,3,5,6,8,9,10,11\}$ |
| $\ell^7(4) = 1601$ | $\theta^7(4) = 6$ | $p^7(4) = \{1,4,5,6,8,9,10,11\}$ |
| $\ell^7(5) = 1547$ | $\theta^7(5) = 3$ | $p^7(5) = \{1,3,5,7,8,9,10,11\}$ |
| $\ell^7(6) = 1599$ | $\theta^7(6) = 5$ | $p^7(6) = \{1,5,6,7,8,9,10,11\}$ |
| $\ell^7(7) = 1589$ | $\theta^7(7) = 6$ | $p^7(7) = \{1,5,6,7,8,9,10,11\}$ |
| $\ell^7(8) = 1122$ | $\theta^7(8) = 8$ | $p^7(8) = \{1,4,8,9,10,11\}$ |
| $\ell^7(9) = 1336$ | $\theta^7(9) = 9$ | $p^7(9) = \{1,2,3,7,8,9,11\}$ |
| $\ell^7(10) = 1575$ | $\theta^7(10) = 9$ | $p^7(10) = \{1,2,3,7,8,9,10,11\}$ |
| $\ell^7(11) = 872$ | $\theta^7(11) = 11$ | $p^7(11) = \{1,4,9,10,11\}$ |

Step (3) $k < 9$, continue

Step (4) $S = \{2,3,4,5,6,7,10\}$

Step (5) $k = 7$

Step (2) $R(S) = \{1,2,3,4,5,6,7,9,11\}$

$\ell^8(2) = 1789$ $\theta^8(2) = 3$ $p^8(2) = \{1,2,3,5,6,8,9,10,11\}$

$\ell^8(3) = 1796$ $\theta^8(3) = 10$ $p^8(3) = \{1,3,4,6,7,8,9,10,11\}$

$\ell^8(4) = 1801$ $\theta^8(4) = 6$ $p^8(4) = \{1,4,5,6,7,8,9,10,11\}$

$\ell^8(5) = 1797$ $\theta^8(5) = 10$ $p^8(5) = \{1,2,3,5,7,8,9,10,11\}$

$\ell^8(6) = 1772$ $\theta^8(6) = 5$ $p^8(6) = \{1,3,5,6,7,8,9,10,11\}$

$\ell^8(7) = 1704$ $\theta^8(7) = 3$ $p^8(7) = \{1,3,5,6,7,8,9,10,11\}$

$\ell^8(8) = 1122$ $\theta^8(8) = 8$ $p^8(8) = \{1,4,8,9,10,11\}$

$\ell^8(9) = 1336$ $\theta^8(9) = 9$ $p^8(9) = \{1,4,6,7,8,9,11\}$

$\ell^8(10) = 1586$ $\theta^8(10) = 10$ $p^8(10) = \{1,2,3,7,8,9,10,11\}$

$\ell^8(11) = 872$ $\theta^8(11) = 11$ $p^8(11) = \{1,4,9,10,11\}$

Step (3) $k < 9$, continue

Step (4) $S = \{2,3,4,5,6,7\}$

Step (5) $k = 8$

Step (2) $R(S) = \{1,2,3,4,5,6,7,9,11\}$

$\ell^9(2) = 2006$ $\theta^9(2) = 3$ $p^9(2) = \{1,2,3,4,6,7,8,9,10,11\}$

$\ell^9(3) = 1796$ $\theta^9(3) = 3$ $p^9(3) = \{1,3,4,6,7,8,9,10,11\}$

$\ell^9(4) = 2224$ $\theta^9(4) = 6$ $p^9(4) = \{1,3,4,5,6,7,8,9,10,11\}$

$\ell^9(5) = 1981$ $\theta^9(5) = 3$ $p^9(5) = \{1,3,4,5,6,7,8,9,10,11\}$

$\ell^9(6) = 2022$ $\theta^9(6) = 5$ $p^9(6) = \{1,2,3,5,6,7,8,9,10,11\}$

$\ell^9(7) = 1710$ $\theta^9(7) = 7$ $p^9(7) = \{1,3,5,6,7,8,9,10,11\}$

$\ell^9(8) = 1122$ $\theta^9(8) = 8$ $p^9(8) = \{1,4,8,9,10,11\}$

$\ell^9(9) = 1336$ $\theta^9(9) = 9$ $p^9(9) = \{1,4,6,7,8,9,11\}$

$\ell^9(10) = 1586$ $\theta^9(10) = 10$ $p^9(10) = \{1,4,6,7,8,9,10,11\}$

$\ell^9(11) = 872$ $\theta^9(11) = 11$ $p^9(11) = \{1,4,9,10,11\}$

Step (3) $k < 9$, continue

Step (4) $S = \{2,4,5,6\}$

Step (5) $k = 9$

Step (2) $R(S) = \{1,3,4,5,6,7,9,11\}$

$\ell^{10}(2) = 2006$ $\theta^{10}(2) = 2$ $p^{10}(2) = \{1,2,3,4,6,7,8,9,10,11\}$

$\ell^{10}(3) = 1796$ $\theta^{10}(3) = 3$ $p^{10}(3) = \{1,3,4,6,7,8,9,10,11\}$

$\ell^{10}(4) = 2224$ $\theta^{10}(4) = 6$ $p^{10}(4) = \{1,2,3,4,5,6,7,8,9,10,11\}$

$\ell^{10}(5) = 1981$ $\theta^{10}(5) = 5$ $p^{10}(5) = \{1,3,4,5,6,7,8,9,10,11\}$

$\ell^{10}(6) = 2022$ $\theta^{10}(6) = 6$ $p^{10}(6) = \{1,2,3,5,6,7,8,9,10,11\}$

$\ell^{10}(7) = 1710$ $\theta^{10}(7) = 7$ $p^{10}(7) = \{1,3,5,6,7,8,9,10,11\}$

$\ell^{10}(8) = 1122$ $\theta^{10}(8) = 8$ $p^{10}(8) = \{1,4,8,9,10,11\}$

$\ell^{10}(9) = 1136$ $\theta^{10}(9) = 9$ $p^{10}(9) = \{1,4,6,7,8,9,11\}$

$\ell^{10}(10) = 1586$ $\theta^{10}(10) = 10$ $p^{10}(10) = \{1,4,6,7,8,9,10,11\}$

$\ell^{10}(11) = 872$ $\theta^{10}(11) = 11$ $p^{10}(11) = \{1,4,9,10,11\}$

Step (3) $k = 9$, stop.

The best achievable tour is obtained as (1-2-3-11-8-7-9-10-5-6-4-1) with the cost 371. Note that, this solution is the same as the one found by the first algorithm. The tours produced for the MDVRP are, therefore, (Figure 5.7a)

Tour 1 = (8-7-8)

Tour 2 = (9-5-6-4-1-2-3-9)

Total cost = 371.

## 5.4 APPLICATION OF ALGORITHM IV

The first step of the fourth algorithm is to determine the convex hull or a partial tour for the problem in order to start the node insertion process. For problems that are not defined in two-dimensional space, however, the problem of determining the convex hull is very difficult or even impossible. Although the original problem (i.e. the MDVRP) is defined in the Euclidean space, the structure of the problem is changed by the transformation. In other words, the transformed matrix does not represent a problem in the Euclidean space anymore. Neither is the triangle inequality satisfied. Consequently, it is not possible to determine the convex hull of the problem since it does not exist. On the other hand, we have do determine a partial tour to start with. A reasonable subtour is T = (8-7-9-10-1-2-3-11-8) and will be used as the starting point (Figure 5.6a).

We will use the cost matrix given in Table 5.6. That is, the cost matrix with all elements $C'_{ij}$ subtracted from a large number L = 250 will be used rather than using the original transformed matrix. The algorithm proceeds as follows:

First, a list for the arcs in T is prepared. The list is given in Table 5.7. Note that, T does not cover all the nodes and the set of candidate nodes to be inserted is {4,5,6}. Since the maximum height in the list corresponds to arc (3,11) with a value of 171.59, node 5 is inserted between nodes 3 and 11. The resultant tour (8-7-9-10-1-2-3-5-11-8) is shown in Figure 5.6b. At this stage, T covers nine nodes and the set of remaining nodes is {4,6}. The list is updated as shown in

Table 5.8. Note that there are three arcs whose end nodes allow the insertion of another node in between. Since, the maximum height

Table 5.7 - List for the arcs in T in the first step

| Starting node | Ending node | Candidate node | Height |
|---|---|---|---|
| 8 | 7 | 6 | 160.83 |
| 7 | 9 | 6 | 160.83 |
| 9 | 10 | - | - |
| 10 | 1 | 5 | 169.98 |
| 1 | 2 | - | - |
| 2 | 3 | - | - |
| 3 | 11 | 5 | 171.59 * |
| 11 | 8 | - | - |

Table 5.8 - List for the arcs in T in the second step

| Starting node | Ending node | Candidate node | Height |
|---|---|---|---|
| 8 | 7 | 6 | 160.83 |
| 7 | 9 | 6 | 160.83 |
| 9 | 10 | - | - |
| 10 | 1 | - | - |
| 1 | 2 | - | - |
| 2 | 3 | - | - |
| 3 | 5 | 6 | 174.72* |
| 5 | 11 | - | - |
| 11 | 8 | - | - |

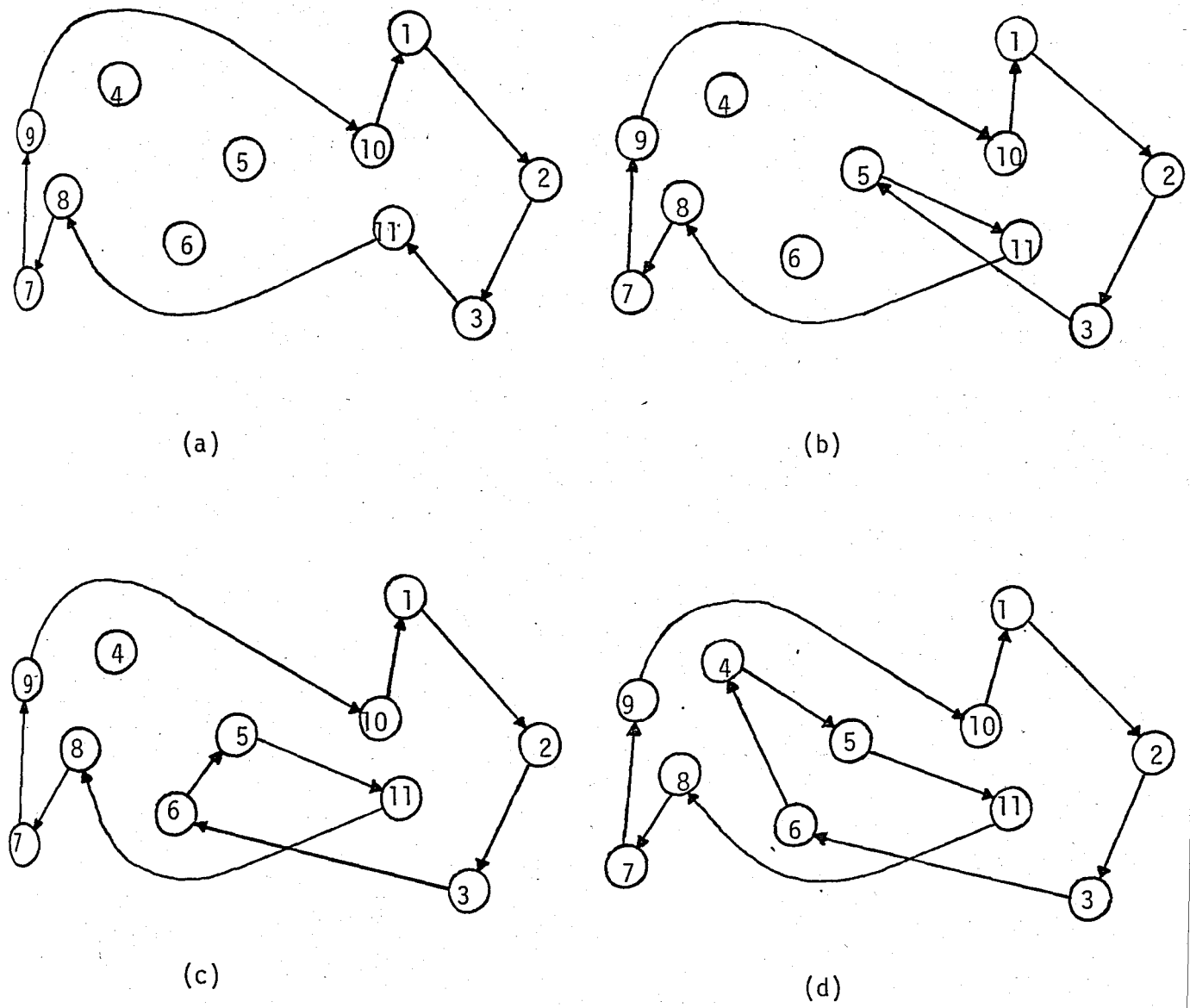



Figure 5.6 - Stages of the node insertion process

corresponds to arc (5,1) with a value of 174.72 we choose arc (5,1) so that node 6 is inserted between nodes 5 and 11. The new tour is T = (8-7-9-10-1-2-3-6-5-11-8) (Figure 6.5c). The number of nodes in T is still less than 11 and only one node, namely node 4, remains to be sequenced. At this point, the list for the arcs in T is as

Table 5.9 - List for the arcs in T in the third step

| Starting node | Ending node | Candidate node | Height |
|---|---|---|---|
| 8 | 7 | - | - |
| 7 | 9 | - | - |
| 9 | 10 | - | - |
| 10 | 1 | - | - |
| 1 | 2 | - | - |
| 2 | 3 | - | - |
| 3 | 6 | - | - |
| 6 | 5 | 4 | 170.73 |
| 5 | 11 | - | - |
| 11 | 8 | - | - |

given in Table 5.9. The list indicates that we do not have much choice. Consequently, being the only location in the sequence node 4 is inserted between nodes 5 and 6. The final tour is obtained as (8-7-9-10-1-2-3-6-4-5-11-8) (Figure 5.6d). The cost of the tour is 372. Using the back transformation once again this tour is subdivided into two tours as follows (Figure 5.7c):

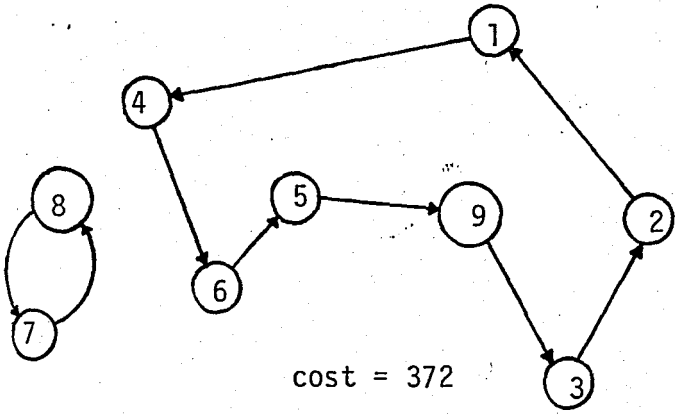
Tour 1 = (8-7-8)

Tour 2 = (9-1-2-3-6-4-5-9)

Total cost = 372

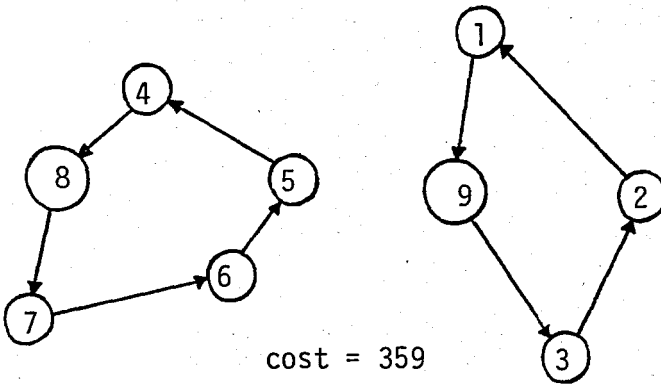
Note that, the best tour among the ones shown in Figure 5.7 is obtained by applying algorithm II and that other tours have the same objective function value. The solution obtained by using algorithm II

is the optimal solution to the problem at the same time.



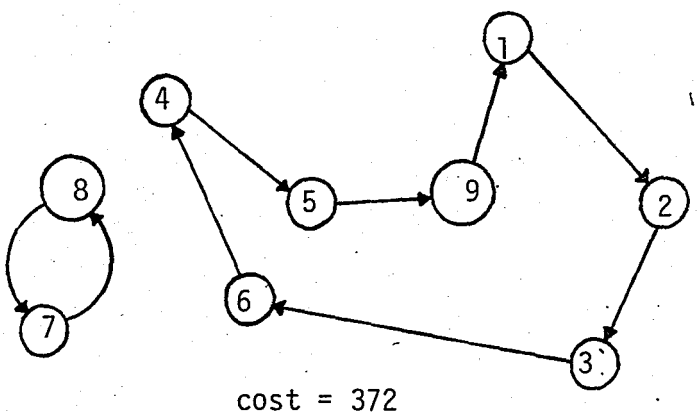

cost = 372

(a)




cost = 359

(b)




cost = 372

(c)

Figure 5.7 - Solutions to the MDVRP

## 5.5 COMPUTATIONAL RESULTS

Considering the difficulties associated with the second and the fourth algorithms, experiments were conducted on the first and the third algorithms. Six complete Euclidean problems were generated on the unit square and the algorithms were applied to them. Actually, the aim in conducting these experiments was to show that solving the TSP on transformed

matrices requires less computation time than it is required for solving the TSP on matrices representing complete graphs of the same size. This is obvious since the rows and columns corresponding to arcs connecting the generated arrival and departure nodes to each other are full of infinities. As a result, there is no need to search on these rows and columns since those arcs have to appear in any feasible solution. The results are indicated in Table 5.10. A careful analysis of Table 5.10

Table 5.10 - Computational results for the MDVRP

| n | No. of vehicles | No. of depots | Total No. of nodes | Algorithm I | | Algorithm III | |
|---|---|---|---|---|---|---|---|
| | | | | Cost | CPU | Cost | CPU |
| 10 | 2 | 2 | 12 | 313 | 2.522 | 284 | 0.913 |
| 20 | 4 | 2 | 26 | 407 | 18.727 | 411 | 4.223 |
| 30 | 4 | 3 | 35 | 534 | 50.666 | 587 | 7.993 |
| 40 | 4 | 4 | 44 | 565 | 90.660 | 587 | 12.444 |
| 50 | 4 | 4 | 54 | 605 | 140.540 | 679 | 21.045 |
| 60 | 7 | 6 | 70 | 749 | 298.361 | 779 | 39.231 |

reveals that for problems of small size, i.e. 10-30 nodes, the computation time for solving an MDVRP seems to be greater than the time required for solving a TSP on a complete graph of the same size as compared with the results given in Chapter 3. However, it should be noted that these figures include the time needed for transforming the MDVRP to an equivalent TSP first, and then making a back transformation after the TSP is solved. Actually, the time needed to perform this procedure

grows linearly with m (i.e. the number of vehicles) and looses its effect as the problem size, n, increases in comparison with m since the algorithms themselves require computation times of order $O(n^3)$. Note that, here n is the number of demand nodes plus two times the number of vehicles. This fact becomes more explicit when the computation times of problems with more than 50 nodes are compared.

# VI. CONCLUSIONS AND EXTENSIONS

Several algorithms developed for solving the TSP are presented in this thesis. First, a literature survey is made on the existing algorithms for solving the TSP in order to give an insight to the various techniques which have proven to be of value up to date. A computational study has been conducted on the new algorithms. We have shown that the algorithms are at least as well as the existing algorithms belonging to the same general class of heuristic procedures available in literature.

The methods used here in solving the TSP were based upon heuristic principles believed to be of general applicability. In dealing with np-complete problems such as the TSP for which an efficient algorithm is unavailable, the general approach is to develop a technique by which near optimum solutions can be obtained very fast. In general, to work on refinement techniques to obtain the best solution has been accepted to be,if not entirely hopeless, time consuming. Instead, much effort is spent on finding the best of a set of good locally optimal solutions which will be close enough to the global optimal solution so as to offer a satisfactory answer in most cases.

In the first algorithm, a tour construction technique is used by the aid of reducing the cost matrix. The effect of reduction is felt in several ways. The most obvious is a considerable decrease in running

time as a result of making the Hamiltonian circuit search on a very small number of arcs. The reduction is based on the existence of a Hamiltonian circuit. Thus, the domain of the search is restricted substantially. This is done even with the possibility that the optimal solution may be lost in the process.

As mentioned earlier, (strong) connectedness and after removing a node from the graph, unilateral connectedness are necessary for the existence of a Hamiltonian circuit but not sufficient. In fact, a feasible solution may not be obtained even though the necessary conditions are satisfied. Then, Little's branch and bound algorithm is applied partially to the resultant cost matrix until a feasible solution is obtained. In other words, the subgraph that has been constructed in the first part of the algorithm is not considered anymore. As an extension of this work, however, a means of further reducing the resultant cost matrix may be investigated. It has been observed that certain arcs appear in all the paths that have been found in the searching process. Therefore, since much of the time spent by the procedure thereafter is essentially a repetition of the previous work, this information may be used to guide further search and reduction and therefore result in saving computational effort. As a result, a decomposition of the cost matrix may be possible. Such a decomposition would not only decrease the size of the matrix being manipulated but also direct the search to find a feasible solution as fast as possible.

For symmetric cost matrices, the reduction procedure may further be improved. That is, a significant reduction in computation effort can be achieved by taking advantage of the fact that the graph is undirected.

Eventually, the effort required for searching the whole matrix can be halved since half of the symmetric matrix contains sufficient information for the whole problem.

Considering the second algorithm, further work may be based on finding all the multiple locally optimal solutions at the end of each iteration so that an optimum or a near optimum solution is not ignored. As mentioned before, the omission of a feasible solution in rank tends to increase the computation effort disproportionally and is not preferable. Another means of extending the study is to find a way of making more than one nonbasic variable which has been removed from the basis previously enter into the basis simultaneously. One possible way of achieving this objective, however, is to consider negative penalties and costs during the calculations with the condition that the final objective function value is positive. This is necessary since a negative objective function value means a decline in the process. On the other hand, attention should be directed to the trade off between the increase in the computation effort and the maximum improvement obtained at each step.

An extension to the third algorithm would be the determination of the root node with which the algorithm starts. It is observed that starting from different nodes yields different solutions. One way of dealing with this fact is, of course, to repeat the algorithm for each possible starting node and take the least cost solution as the best achievable one. However, one should note that the number of comparisons and calculations will be multiplied by n. A second and easily applicable extension is first to generate a tour with regard to this algorithm and then to test each node on the tour between each consecutive pair on the

tour to see if such a change in the sequence will lower the cost. This is the simplest case of the processes which are referred to as "tours optimal relative to insertion and inversion" [49]. The process continues until no improvement is possible relative to insertion and inversion. Needless to say, an extra computation effort would be required in this case. However, one should be careful in directing his attention to finding improvements with a minimum amount of computation rather than to making the maximum improvement possible.

The insertion and inversion process can also be applied to the tours generated by the fourth algorithm. To achieve a tour optimal relative to the one produced by the algorithm, the testing process must be started from the beginning each time the tour is improved. In addition to improving the tours produced, extended work may be based on defining the convex hull since the identification of the convex hull plays an important role in forming the final sequence. Overall, the extensions should be evaluated by considering the trade-offs between effectiveness and computation effort. The question "How much can the computation effort be decreased by sacrificing some effectiveness" should always be kept in mind.

The second part of the thesis is focused on the application of the algorithms to a special routing problem, namely the multi-depot vehicle routing problem. The relevance between the TSP and the VRPs is emphasized by first considering the TSP as the simplest VRP and then progressing from the simplest to the more complex. As a result, the MTSP is considered first and the MDVRP next. In each problem, we describe the constraints which are added to the previous problem in order to present the steps which lead to the more complex.

The fact that the MTSP can be transformed to an equivalent TSP has been studied extensively. However, the MDVRP which is an extension of the MTSP has not received that much attention. On the other hand, we have shown that the MDVRP can also be transformed to an equivalent TSP. Heuristic methods presented in the relevant literature do not consider this possibility. Moreover, no efficient algorithm has been developed for solving even small size MDVRPs efficiently. However, exact algorithms developed for solving the TSP have shown a considerable progress in comparison with methods developed for other np-complete problems. Therefore, exact solution methods for solving the TSP can be used as a tool for solving MDVRPs of reasonable sizes. MDVRPs of large size, however, can be solved by using efficient heuristics developed for the TSP.

The application of the heuristic algorithms presented in the thesis to the MDVRP showed that, in general, the algorithms require less computation time than it is required for solving the TSP on a complete graph of the same size. This is an important result, since it implies that heuristics developed for the TSP can be applied to the MDVRP more efficiently.

An important extension which is a promising area for further work is the application of the fourth algorithm presented in the thesis to the MDVRP directly in a modified version. With the additional restriction that all of the vehicles in the depot will be used, the algorithm may be used to produce independent partial tours separately. A problem arises in determining the starting partial tours. One way of dealing with this difficulty is to form subtours containing the two arcs which join the nearest nodes to the depots. However, some other means of dealing with this problem can be found.

# APPENDIX A

```
ROGRAM THESIS( INPUT,OUTPUT );

(*************************************************************)
 THIS PROGRAM IS PREPEARED BY YASEF IOVYA,  JULY, 1983       *)
 INSTITUTE FOR GRADUATE STUDIES IN SCIENCE AND ENGINEERING   *)
 BOGAZICI UNIVERSITY, ISTANBUL                               *)
                                                             *)
 THIS PROGRAM IS THE CODE OF 4 DIFFERENT HEURISTIC ALGORITHMS *)
 FOR SOLVING THE TRAVELING SALESMAN PROBLEM (TSP). IN ADDITION *)
 GIVEN A DISTANCE MATRIX ASSOCIATED WITH A MULTIPLE - DEPOT -  *)
 VEHICLE ROUTING PROBLEM (MDVRP) , THE PROGRAM CREATES A      *)
 TRANSFORMED MATRIX SO THAT THE SOLUTION TO THE MDVRP CAN BE  *)
 OBTAINED BY SOLVING THE TSP ON THE TRANSFORMED MATRIX.....   *)
                                                             *)
(*************************************************************)

ONST ND = 70; ND1 = 71; NDN = 2000;

YPE MATRIX = ARRAY(.1..ND,1..ND.) OF INTEGER;
     ARRY = ARRAY(.1..ND.) OF INTEGER;

AR F,C : MATRIX;
    VN,NODE,HC,VNU : ARRY;
    DEMAND,DEPOTS : SET OF 1..ND;
    N1,INF,TSP,VDP,NDEM,NDEP,IDD,J,TNV : INTEGER;
    I,K,N,L,M,MM,L1,L2,VU,KM : INTEGER;

ROCEDURE PRINT( VAR C : MATRIX;
                VAR N : INTEGER );

(*************************************************************)
 THIS PROCEDURE OUTPUTS ANY N X N SQUARE MATRIX              *)
(*************************************************************)

AR I,J,J1,J2 : INTEGER;

EGIN WRITELN;
  FOR I:=1 TO N DO
  BEGIN WRITELN; J1:=0; J2:=0;
    REPEAT WRITELN; WRITE(,   ROW,,I:3,,  ,);
      J1:=J2+1; J2:=J2+20;
      IF J2 > N THEN J2:=N;
      FOR J:=J1 TO J2 DO WRITE(C(.I,J.):5)
    UNTIL J2 = N
  END
ND;

ROCEDURE YTSP( VAR C : MATRIX;
               VAR MHC : ARRY;
               VAR N,INF : INTEGER );

(*************************************************************)
 THIS PROCEDURE FINDS A HEURISTIC SOLUTION TO THE TSP BY      *)
 CREATING A SUBGRAPH GO WHICH IS COMPRISED BY ARCS WITH ZERO  *)
 COSTS AS A RESULT OF REDUCING THE COST MATRIX. THE REDUCTION *)
 IS CONTINUED UNTIL A HAMILTONIAN CIRCUIT EXISTS IN GO.       *)
(*************************************************************)

YPE ARR1 = ARRAY(.1..NDN.) OF INTEGER;
     ARR2 = ARRAY(.1..ND1.) OF INTEGER;
     ARR3 = ARRAY(.1..ND.) OF INTEGER;
     NODES = SET OF 1..ND;
     ARR4 = ARRAY(.1..ND.) OF NODES;

AR FAR,BAR : ARR1;
    PB,PF : ARR2;
    OD,ID,HC,CAR,PA : ARR3;
    R,S1,S2 : NODES;
    D : MATRIX;
    RR,FR : ARR4;
    I,J,NN,M,NI,CON,L,K,MIN,MAXM,NS,NR,RN,ALL,FF,COST,
    MAX,MCOST,I1,I2,COSTA,MCOSTA : INTEGER;

ROCEDURE HAMIL( VAR FAR,BAR : ARR1;
                VAR PF,PB : ARR2;
                VAR OD,ID : ARR3;
                VAR N,M,RN,ALL,FF : INTEGER);

(*************************************************************)
 THIS PROCEDURE FINDS ONE OR MORE HAMILTONIAN CIRCUITS IN A   *)
 DIRECTED GRAPH BY AN ENUMERATIVE METHOD                      *)
```

```
(*************************************************************)

ABEL 100,200,300,400,500,600;
AR KLI,CI,P,TOR,HC : ARRAY(.1..ND.) OF INTEGER;
   T,NP1,MP1,K1,K,J,J1,J2,L,JL,PER,I1,I2,JJ,LL,JS,
   L1,L2,DD,S,IF,KKK,IP,JA,K2,IFF,II,KJ,MN,IL,II1,II2 : INTEGER;

ROCEDURE PATH( VAR T,J,PER,I1,I2 : INTEGER);

(*************************************************************)
 THIS PROCEDURE FINDS THE LARGEST PATH THAT CAN BE FORMED BY   *)
 THE IMPLIED ARCS AND THAT CONTAINS ARC (T,J). THE STARTING    *)
 NODE AND THE ENDING NODE OF THE PATH ARE ALSO SPECIFIED...    *)
(*************************************************************)

AR L,KK,L1,JA : INTEGER;

EGIN PER:=0; L:=1; I1:=I;
 WHILE CI(.I1.) <> 0 DO
 BEGIN I1:=CI(.I1.); L:=L+1
 END;
 I2:=J; L:=L+1;
 WHILE KLI(.I2.) <> 0 DO
 BEGIN I2:=-KLI(.I2.)+(KLI(.I2.) DIV NP1)*NP1; L:=L+1
 END;
 IF L = N THEN
 BEGIN KK:=-K*NP1; L:=PF(.I2.); L1:=0;
   REPEAT L:=L+1;
     IF FAR(.L.) = T1 THEN L1:=1 ELSE
     IF (FAR(.L.) < 0) AND (KK-FAR(.L.) = T1) THEN L1:=1
   UNTIL (L1 = 1) OR (L = PF(.I2+1.));
   IF L1 <> 1 THEN PER:=-1 ELSE
   BEGIN PER:=1; CI(.J.):=I;
     CI(.T1.):=T2; HC(.N.):=CI(.RN.); L:=N-1;
     WHILE L <> K DO
     BEGIN JA:=HC(.L+1.); HC(.L.):=CI(.JA.); L:=L-1
     END;
     CI(.T1.):=0; CI(.J.):=0
   END
 END
ND;

ROCEDURE TUPD( VAR TA,IB,L,K1 : INTEGER;
               VAR A1,A2 : ARR1;
               VAR P1,P2 : ARR2;
               VAR D1,D2 : ARR3 );

(*************************************************************)
* THIS PROCEDURE REMOVES ALL THE ARCS EITHER EMANATING FROM OR  *)
* TERMINATING AT IB.                                            *)
(*************************************************************)

AR M,IAR,J : INTEGER;

EGIN M:=P1(.IB.);
 REPEAT M:=M+1; IAR:=A1(.M.);
   IF IAR > 0 THEN
   BEGIN
     IF D2(.IAR.) <> 1 THEN
     BEGIN J:=P2(.IAR.);
       REPEAT J:=J+1
       UNTIL (A2(.J.) = IB) OR (J = P2(.IAR+1.))
     END ELSE
     IF  IAR = TA THEN J:=L ELSE TA:=0;
     IF TA <> 0 THEN
     BEGIN A2(.J.):=K1-A2(.J.);
       D2(.IAR.):=D2(.IAR.)-1;
       A1(.M.):=K1-IAR;
       D1(.IB.):=D1(.IB.)-1
     END
   END
 UNTIL (TA = 0) OR (M = P1(.IB+1.))
ND;

ROCEDURE FUPD( VAR A1,A2 : ARR1;
               VAR P1,P2 : ARR2;
               VAR D1,D2 : ARR3;
               VAR T,K1 : INTEGER );

(*************************************************************)
* THIS PROCEDURE PERFORMS THE FORWARD UPDATING PHASE...         *)
(*************************************************************)
```

```
AR J,IA,L : INTEGER;

EGIN
  FOR J:=P1(.I.)+1 TO P1(.I+1.) DO
  IF A1(.J.) > 0 THEN
  BEGIN IA:=A1(.J.); L:=P2(.IA.);
    REPEAT L:=L+1
    UNTIL (A2(.L.) = I) OR (L = P2(.IA+1.));
    D2(.IA.):=D2(.IA.)-1;
    A2(.L.):=K1-A2(.L.); A1(.J.):=K1-IA
  END;
  D1(.I.):=0
ND;

ROCEDURE BUPD( VAR A1,A2 : ARR1;
               VAR P1,P2 : ARR2;
               VAR D1,D2 : ARR3;
               VAR I,K1,K2 : INTEGER );

(*****************************************************)
* THIS PROCEDURE PERFORMS THE BACKWARD UPDATING PHASE.... *)
(*****************************************************)

AR L,IA,J : INTEGER;

EGIN
  FOR L:=P1(.I.)+1 TO P1(.I+1.) DO
  IF (A1(.L.) <= K1) AND (A1(.L.) >= K2) THEN
  BEGIN IA:=K1-A1(.L.); A1(.L.):=IA;
    D1(.I.):=D1(.I.)+1; J:=P2(.IA.);
    REPEAT J:=J+1
    UNTIL (K1-A2(.J.) = I) OR (J = P2(.IA+1.));
    A2(.J.):=I; D2(.IA.):=D2(.IA.)+1
  END
ND;

ROCEDURE BARC( VAR IA,IB,K1,JJ,LL : INTEGER );

(*****************************************************)
* THIS PROCEDURE TRIES TO REMOVE ARC (IA,IB) FROM THE GRAPH.  *)
* THE ARC MAY NOT BE REMOVED DUE TO THE FACT THAT EITHER IT   *)
* IS NOT IN THE GRAPH OR ITS REMOVAL PREVENTS THE EXISTENCE   *)
* OF A HAMILTONIAN CIRCUIT....                                *)
(*****************************************************)

AR IC : INTEGER;

EGIN JJ:=PF(.IA.); IC:=0;
  REPEAT JJ:=JJ+1;
    IF ( BAR(.JJ.) > 0) AND (BAR(.JJ.) = IB) THEN
    BEGIN LL:=PB(.IB.);
      REPEAT LL:=LL+1
      UNTIL (BAR(.LL.) = IA) OR (LL = PB(.IB+1.));
      IF BAR(.LL.) = IA THEN IC:=1
    END
  UNTIL (IC = 1) OR (JJ = PF(.IA+1.));
  IF IC = 0 THEN JJ:=0 ELSE
  IF (OD(.IA.) = 1) OR (ID(.IB.) = 1) THEN JJ:=-1 ELSE
  BEGIN BAR(.JJ.):=K1-IB;
    OD(.IA.):=OD(.IA.)-1;
    BAR(.LL.):=K1-IA;
    ID(.IB.):=ID(.IB.)-1
  END
ND;

(*****************)
* STEP 0 INITIALIZE *)
(*****************)

EGIN FF:=0;
  FOR I:=1 TO N DO
  BEGIN CT(.I.):=0; KLI(.I.):=0; P(.I.):=1; IOR(.I.):=0
  END;
  NP1:=N+1; MP1:=M+1;
  K1:=-NP1; K:=1; HC(.1.):=RN;

(*********************************)
* STEP 1 SEARCH FOR IMPLIED ARCS *)
(*********************************)

  J:=1; J1:=0; J2:=0;
```

```
    REPEAT IF OD(.J.) = 1 THEN J1:=1
      ELSE IF ID(.J.) = 1 THEN J2:=1
      ELSE J:=J+1
    UNTIL (J > N) OR (J1 = 1) OR (J2 = 1);
    IF J > N THEN GOTO 200;
    IF J1 = 1 THEN
    BEGIN L:=PF(.J.);
      REPEAT L:=L+1
      UNTIL (FAR(.L.) > 0) OR (L = PF(.J+1.));
      JL:=FAR(.L.);
      PATH(J,JL,PER,I1,I2);
      IF PER = 0 THEN
      BEGIN KLI(.J.):=K1-JL; CI(.JL.):=J;
        IUPD(J,JL,L,K1,FAR,BAR,PF,PB,ID,OD);
        IF J = 0 THEN GOTO 400 ELSE
        BEGIN RARC(I2,I1,K1,JJ,LL);
          IF JJ = -1 THEN GOTO 400
          ELSE GOTO 100
        END
      END ELSE
      IF PER = -1 THEN GOTO 400 ELSE
      BEGIN K:=K+1; GOTO 500
      END
    END;
    IF J2 = 1 THEN
    BEGIN L:=PB(.J.);
      REPEAT L:=L+1
      UNTIL (BAR(.L.) > 0) OR (L = PB(.J+1.));
      JL:=BAR(.L.);
      PATH(JL,J,PER,I1,I2);
      IF PER = 0 THEN
      BEGIN KLI(.JL.):=K1-J; CI(.J.):=JL;
        IUPD(J,JL,L,K1,FAR,BAR,PF,PB,OD,ID);
        IF J = 0 THEN GOTO 400 ELSE
        BEGIN RARC(I2,I1,K1,JJ,LL);
          IF JJ = -1 THEN GOTO 400
          ELSE GOTO 100
        END
      END ELSE
      IF PER = -1 THEN GOTO 400 ELSE
      BEGIN I:=HC(.K.); K:=K+1; GOTO 500
      END
    END;

  (*****************************)
  (* STEP 2  ADD IMPLIED ARCS TO HC  *)
  (*****************************)

200 : REPEAT I:=HC(.K.);
      IF KLI(.I.) <> 0 THEN
      BEGIN JS:=-KLI(.I.)+(KLI(.I.) DIV NP1)*NP1;
        IF JS = RN THEN GOTO 400 ELSE
        BEGIN K:=K+1; HC(.K.):=JS
        END
      END ELSE GOTO 300
    UNTIL K = N;
    IF KLI(.JS.) < 0 THEN GOTO 500
    ELSE GOTO 400;

  (*****************)
  (* STEP 3  BRANCH  *)
  (*****************)

300 : L1:=PF(.I.)+P(.I.); L2:=PF(.I+1.);
    IF L1 > L2 THEN GOTO 400 ELSE
    BEGIN OD:=N*SGN(N); J1:=0; J2:=0;
      FOR J:=L1 TO L2 DO
      BEGIN JL:=FAR(.J.);
        IF JL > 0 THEN
        BEGIN KKK:=1;
          IF OD(.JL.) > 0 THEN
          BEGIN S:=OD(.JL.)*N+ID(.JL.);
            IF OD(.JL.) > ID(.JL.) THEN
            S:=ID(.JL.)*N+OD(.JL.)
          END ELSE
          IF (KLI(.JL.) <> 0) AND (JL <> RN) THEN
          BEGIN IE:=JL;
            REPEAT IE:=-KLI(.IE.)+(KLI(.IE.) DIV NP1)*NP1
            UNTIL KLI(.IE.) = 0;
            IF ID(.JL.) < OD(.IE.) THEN
            S:=ID(.JL.)*N+OD(.IE.) ELSE
            S:=OD(.IE.)*N+ID(.JL.)
```

```
          END ELSE KKK:=0;
          IF KKK = 1 THEN
          BEGIN
            IF OD > S THEN
            BEGIN OD:=S; IP:=J
            END;
            IF J1 = 0 THEN J1:=J ELSE
            IF J2 = 0 THEN J2:=J
          END
        END
      END;
      IF J1 = 0 THEN GOTO 400 ELSE
      BEGIN JL:=FAR(.IP.);
        FAR(.IP.):=FAR(.J1.); FAR(.J1.):=JL;
        IF J2 = 0 THEN J2:=PF(.I+1.)+1;
        P(.I.):=J2-PF(.I.); K:=K+1;
        HC(.K.):=JL; K1:=-K*MP1;
        BUPD(FAR,BAR,PF,PB,OD,ID,I,K1);
        BUPD(BAR,FAR,PB,PF,ID,OD,JL,K1);
        TOR(.K.):=0;
        BARC(JL,RN,K1,JJ,LL);
        IF JJ = -1 THEN GOTO 400 ELSE
        IF JJ <> 0 THEN TOR(.K.):=JJ*MP1+LL;
        GOTO 100
      END
    END;

  (*****************)
  (* STEP 4  BACKTRACK  *)
  (*****************)

400 : IF K > 1 THEN
    REPEAT JA:=HC(.K.);
      P(.JA.):=1; JA:=HC(.K-1.);
      IF KLI(.JA.) <> 0 THEN K:=K-1
    UNTIL (KLI(.JA.) = 0) OR (K = 1);
    IF K > 1 THEN
    BEGIN K1:=-K*MP1; K2:=K1-MP1;
      I:=HC(.K-1.); IFF:=0;
      FOR J:=1 TO N DO
      IF (KLI(.J.) <= K1) AND (KLI(.J.) >= K2) THEN
      BEGIN JA:=K1-KLI(.J.);
        KLI(.J.):=0; IFF:=1;
        CI(.JA.):=0
      END;
      IF IFF = 1 THEN
      FOR J:=1 TO N DO
      BEGIN L1:=PF(.J.)+1; L2:=PF(.J+1.);
        FOR L:=L1 TO L2 DO
        BEGIN JL:=FAR(.L.);
          IF (JL <= K1) AND (JL >= K2) THEN
          BEGIN JL:=K1-JL;
            FAR(.L.):=JL; OD(.J.):=OD(.J.)+1;
            LL:=PB(.JL.);
            REPEAT LL:=LL+1
            UNTIL (K1-BAR(.LL.) = J) OR (LL = PB(.JL+1.));
            BAR(.LL.):=J; ID(.JL.):=ID(.JL.)+1
          END
        END
      END ELSE
      BEGIN II:=HC(.K.);
        BUPD(FAR,BAR,PF,PB,OD,ID,I,K1,K2);
        BUPD(BAR,FAR,PB,PF,ID,OD,II,K1,K2);
        IF TOR(.K.) <> 0 THEN
        BEGIN J1:=TOR(.K.) DIV MP1;
          J2:=TOR(.K.)-J1*MP1; FAR(.J1.):=RN;
          JA:=HC(.K.); OD(.JA.):=OD(.JA.)+1;
          BAR(.J2.):=HC(.K.); ID(.RN.):=ID(.RN.)+1
        END
      END;
      K:=K-1; GOTO 300
    END ELSE
    WRITELN; WRITELN;
    WRITELN('  NO HAMILTONIAN CIRCUIT CAN BE FOUND.');
    GOTO 600;

  (*****************************************)
  (* STEP 5  A HAMILTONIAN CIRCUIT IS FOUND  *)
  (*****************************************)

500 : WRITELN; WRITELN;
    WRITELN('  HAMILTONIAN  CIRCUIT :.');
```

```
    KJ:=0; MN:=0; COST:=0;
    REPEAT KJ:=KJ+1; MN:=MN+20; WRITELN(,    ,);
      IF MN > N THEN MN:=N;
      FOR II:=KJ TO MN DO
      BEGIN II1:=HC(.II.);
        IF II <> N THEN II2:=HC(.II+1.) ELSE II2:=HC(.1.);
        COST:=COST+D(.II1,II2.); WRITE(HC(.II.):4)
      END
    UNTIL MN = N;
    WRITELN; WRITELN(,   COST =,,COST:8);
    IF COST < MCOST THEN
    BEGIN MCOST:=COST;
      FOR II:=1 TO N DO MHC(.II.):=HC(.II.)
    END;
    FF:=1;
    IF ALL = 1 THEN
    BEGIN K:=K-1; GOTO 400
    END ELSE GOTO 600;
600 : WRITELN; WRITELN;
    WRITELN(,   END OF HAMILTONIAN CIRCUIT SEARCH,)
  END;

  PROCEDURE FORADJ( VAR NN,M : INTEGER );

  (*****************************************************************)
  (* THIS PROCEDURE CREATES THE FORWARD ADJACENCY ARRAY OF THE      *)
  (* SUBGRAPH GO AND SPECIFIES THE NUMBER OF ARCS M IN GO....      *)
  (*****************************************************************)

  VAR I,J : INTEGER;

  BEGIN M:=0; PF(.1.):=0;
    FOR I:=1 TO N DO ID(.I.):=0;
    FOR I:=1 TO N DO
    BEGIN
      FOR J:=1 TO N DO
      IF C(.I,J.) <= NN THEN
      BEGIN M:=M+1; FAR(.M.):=J; ID(.J.):=ID(.J.)+1
      END;
      PF(.I+1.):=M; OD(.I.):=M-PF(.I.)
    END
  END;

  PROCEDURE BACKAD;

  (*****************************************************************)
  (* THIS PROCEDURE FORMS THE BACKWARD ADJACENCY ARRAY OF GO...     *)
  (*****************************************************************)

  VAR I,J,L,JA : INTEGER;

  BEGIN PB(.1.):=0;
    FOR I:=1 TO N DO
    BEGIN PB(.I+1.):=PB(.I.)+ID(.I.); ID(.I.):=0
    END;
    FOR I:=1 TO N DO
    FOR J:=PF(.I.)+1 TO PF(.I+1.) DO
    BEGIN L:=FAR(.J.);
      ID(.L.):=ID(.L.)+1;
      JA:=PB(.L.)+ID(.L.);
      BAR(.JA.):=I
    END
  END;

  PROCEDURE ROOTNODE( VAR RN : INTEGER );

  (*****************************************************************)
  (* THIS PROCEDURE FINDS THE ROOT NODE WITH WHICH THE              *)
  (* HAMILTONIAN CIRCUIT SEARCH WILL START...                      *)
  (*****************************************************************)

  VAR MAX,MIN,I : INTEGER;

  BEGIN MAX:=ID(.1.); MIN:=OD(.1.); RN:=1;
    FOR I:=2 TO N DO
    IF MAX < ID(.I.) THEN
    BEGIN MAX:=ID(.I.); MIN:=OD(.I.); RN:=I
    END ELSE
    IF MAX = ID(.I.) THEN
    IF MIN > OD(.I.) THEN
    BEGIN MIN:=OD(.I.); RN:=I
    END
```

```
END;

PROCEDURE REDUCE;

(*****************************************************)
(* THIS PROCEDURE REDUCES THE COST MATRIX...         *)
(*****************************************************)

VAR I,J,MIN : INTEGER;

BEGIN  (* REDUCE *)
  FOR I:=1 TO N DO ID(.I.):=0;
  FOR I:=1 TO N DO
  BEGIN MIN:=INF;
    FOR J:=1 TO N DO
    IF C(.I,J.) < MIN THEN MIN:=C(.I,J.);
    FOR J:=1 TO N DO
    IF C(.I,J.) > MIN THEN
    BEGIN IF C(.I,J.) <> INF THEN C(.I,J.):=C(.I,J.)-MIN
    END ELSE
    IF C(.I,J.) = MIN THEN
    BEGIN  ID(.J.):=ID(.J.)+1; C(.I,J.):=0
    END
  END;
  FOR J:=1 TO N DO
  IF ID(.J.) = 0 THEN
  BEGIN MIN:=INF;
    FOR I:=1 TO N DO
    IF C(.I,J.) < MIN THEN MIN:=C(.I,J.);
    FOR I:=1 TO N DO
    IF C(.I,J.) <> INF THEN C(.I,J.):=C(.I,J.)-MIN
  END
END;

PROCEDURE REACH( VAR NI : INTEGER );

(*****************************************************)
(* THIS PROCEDURE FINDS THE NODE SET REACHABLE FROM NODE NI... *)
(*****************************************************)

VAR NP,I,J,K,L : INTEGER;

BEGIN
  FOR NP:=1 TO N DO
  IF NP <> NI THEN
  BEGIN RR(.NP.):=(.NP.);
    FOR I:=1 TO N DO PA(.I.):=0;
    PA(.1.):=NP; L:=1; K:=1;
    REPEAT J:=PA(.L.); I:=PF(.J.);
      REPEAT I:=I+1;
        IF FAR(.I.) <> NI THEN
        IF NOT(FAR(.I.) IN RR(.NP.)) THEN
        BEGIN RR(.NP.):=RR(.NP.)+(.FAR(.I.).); K:=K+1; PA(.K.):=FAR(.I
        END
      UNTIL (I = PF(.J+1.)) OR (K = N-1);
      IF (K < N-1) THEN L:=L+1
    UNTIL (PA(.L.) = 0) OR (K = N-1);
    CAR(.NP.):=K
  END
END;

PROCEDURE LIT( VAR D : MATRIX;
               VAR HC : ARRAY;
               VAR N,INF : INTEGER );

(*****************************************************)
(* THIS PROCEDURE APPLIES LITTLE'S ALGORITHM PARTIALLY...  *)
(*****************************************************)

VAR INFT : ARRAY(.1..ND,1..3.) OF INTEGER;
    TRE : ARRAY(.1..ND,1..3.) OF INTEGER;
    LUP,RUP,CUP : ARRAY(.1..ND.) OF INTEGER;
    ROWW,COLL : SET OF 1..ND;
    F : MATRIX;
    LEVEL,C1,C2,MIN,MING,M,L,MAX,K : INTEGER;
    I,J,MT,MAXT,K1,K2,IE,ID,II,IJ : INTEGER;

PROCEDURE REDROW( VAR C1 : INTEGER );

(*****************************************************)
(* THIS PROCEDURE PERFORMS THE ROW REDUCTION...      *)
(*****************************************************)
```

```
VAR I,J,MIN : INTEGER;

BEGIN C1:=0;
  FOR I:=1 TO N DO
  BEGIN MIN:=0;
    IF NOT(I IN ROWW) THEN
    BEGIN MIN:=INF;
      FOR J:=1 TO N DO
      IF NOT(J IN COLL) THEN
      IF E(.I,J.) < MIN THEN MIN:=E(.I,J.);
      IF MIN <> 0 THEN
      FOR J:=1 TO N DO E(.I,J.):=E(.I,J.)-MIN;
      C1:=C1+MIN
    END
  END
END;

PROCEDURE REDCOL( VAR C2 : INTEGER );

(*****************************************************************)
(* THIS PROCEDURE PERFORMS THE COLUMN REDUCTION...               *)
(*****************************************************************)

VAR I,J,MIN : INTEGER;

BEGIN C2:=0;
  FOR I:=1 TO N DO
  BEGIN MIN:=0;
    IF NOT(I IN COLL) THEN
    BEGIN MIN:=INF;
      FOR J:=1 TO N DO
      IF NOT(J IN ROWW) THEN
      IF E(.J,I.) < MIN THEN MIN:=E(.J,I.);
      IF MIN <> 0 THEN
      FOR J:=1 TO N DO E(.J,I.):=E(.J,I.)-MIN;
      C2:=C2+MIN
    END
  END
END;

BEGIN MING:=INF;
  FOR I:=1 TO N DO
  BEGIN INFT(.I,1.):=INF;
    TRE(.I,1.):=0; TRE(.I,2.):=0;
    FOR J:=1 TO N DO E(.I,J.):=D(.I,J.);
    RUP(.I.):=0; CUP(.I.):=0; LUP(.I.):=0
  END;
  ROWW:=(..); COLL:=(..);
  REPEAT LEVEL:=1;
    REDROW(C1); REDCOL(C2);
    MIN:=C1+C2;
    REPEAT M:=0;
      L:=0; MAX:=0;
      FOR I:=1 TO N DO
      IF NOT(I IN ROWW) THEN
      FOR J:=1 TO N DO
      IF NOT(J IN COLL) THEN
      IF E(.I,J.)=0 THEN
      BEGIN
        IF M = 0 THEN
        BEGIN M:=I; L:=J
        END;
        MT:=INF;
        FOR K:=1 TO N DO
        IF NOT(K IN ROWW) THEN
        IF E(.K,J.) < MT THEN
        IF K <> I THEN MT:=E(.K,J.);
        MAXT:=MT; MT:=INF;
        FOR K:=1 TO N DO
        IF NOT(K IN COLL) THEN
        IF E(.I,K.) < MT THEN
        IF K <> J THEN MT:=E(.I,K.);
        MAXT:=MAXT+MT;
        IF MAXT > MAX THEN
        BEGIN MAX:=MAXT; M:=I; L:=J
        END
      END;
      ROWW:=ROWW+(.M.); COLL:=COLL+(.L.);
      CUP(.L.):=M; RUP(.M.):=L; LUP(.M.):=LEVEL;
      IF (LEVEL < N-1) THEN
      BEGIN K1:=L; K2:=M;
```

```
        WHILE RUP(.K1.) <> 0 DO K1:=RUP(.K1.);
        WHILE CUP(.K2.) <> 0 DO K2:=CUP(.K2.);
        E(.K1,K2.):=INF
      END;
      TRE(.LEVEL,1.):=MIN+MAX;
      REDROW(C1); REDCOL(C2);
      MIN:=MIN+C1+C2;
      TRE(.LEVEL,2.):=MIN;
      LEVEL:=LEVEL+1
    UNTIL (LEVEL > N) OR (MIN >= MING);
    IF MIN < MING THEN
    BEGIN I:=0; K:=1;
      REPEAT I:=I+1; HC(.I.):=K; K:=RUP(.K.)
      UNTIL I = N
    END ELSE
    BEGIN I:=0; IE:=0;
      REPEAT I:=I+1;
        IF TRE(.LEVEL-I,1.) < MIN THEN IE:=1
      UNTIL (IE = 1) OR (I = LEVEL-1);
      IF IE = 1 THEN
      BEGIN LEVEL:=LEVEL-I;
        ROWW:=(..); COLL:=(..);
        FOR I:=1 TO N DO
        BEGIN TRE(.I,1.):=0; TRE(.I,2.):=0;
          FOR J:=1 TO N DO E(.I,J.):=D(.I,J.);
          IF LUP(.I.) = LEVEL THEN
          BEGIN M:=I; L:=RUP(.I.)
          END;
          CUP(.I.):=0; RUP(.I.):=0; LUP(.I.):=0;
        END;
        IP:=0;
        FOR I:=1 TO N DO
        IF INFT(.I,1.) > LEVEL THEN
        BEGIN
          IF IP = 0 THEN
          BEGIN IP:=1;
            INFT(.I,1.):=LEVEL; INFT(.I,2.):=M;
            INFT(.I,3.):=L; E(.M,L.):=INF
          END ELSE INFT(.I,1.):=INF
        END ELSE
        BEGIN II:=INFT(.I,2.);
          IJ:=INFT(.I,3.);
          E(.II,IJ.):=INF
        END
      END
    END
  UNTIL (MIN < MING) OR (IE = 0)
END;

BEGIN MCOST:=INF;
  FOR I:=1 TO N DO
  FOR J:=1 TO N DO D(.I,J.):=C(.I,J.);
  REDUCE; NN:=0;
  FORADJ(NN,M);

(*********************)
(* OBTAIN CONNECTEDNESS *)
(*********************)

  FOR NT:=1 TO N DO
  BEGIN
    FOR I:=1 TO N DO PA(.I.):=0;
    REPEAT PA(.1.):=NI; CON:=0;
      L:=1; R:=(.NI.); K:=1;
      REPEAT J:=PA(.L.); I:=PF(.J.);
        REPEAT I:=I+1;
          IF NOT(FAR(.I.) IN R) THEN
          BEGIN R:=R+(.FAR(.I.).); K:=K+1; PA(.K.):=FAR(.I.)
          END
        UNTIL (I = PF(.J+1.)) OR (K = N);
        IF K = N THEN CON:=1 ELSE L:=L+1
      UNTIL (PA(.L.) = 0) OR (K = N);
      IF CON = 0 THEN
      BEGIN MIN:=INF;
        FOR I:=1 TO N DO
        IF I IN R THEN
        FOR J:=1 TO N DO
        IF NOT(J IN R) THEN
        IF C(.I,J.) < MIN THEN MIN:=C(.I,J.);
        FOR I:=1 TO N DO
        IF I IN R THEN
        FOR J:=1 TO N DO
```

```
        IF NOT(J IN R) THEN
        IF C(.I,J.) <> INF THEN C(.I,J.):=C(.I,J.)-MIN;
        FORADJ(NN,M);
        FOR I:=1 TO N DO PA(.I.):=0
      END
    UNTIL CON = 1
  END;

(*****************************************)
(* OBTAIN UNILATERAL CONNECTEDNESS *)
(*****************************************)

  FOR NI:=1 TO N DO
  BEGIN REACH(NI);
    FOR NS:=1 TO N DO
    IF NS <> NI THEN
    IF (CAR(.NS.) < N-1) THEN
    FOR NR:=1 TO N DO
    IF (NR <> NI) AND (NR <> NS) THEN
    IF (CAR(.NR.) <N-1) THEN
    REPEAT CON:=1;
      IF (NOT(NS IN RR(.NR.))) AND (NOT(NR IN RR(.NS.))) THEN
      BEGIN MIN:=INF; CON:=0;

(*****************************************)
(* THERE IS NO PATH EITHER FROM NS *)
(* TO NR OR VICE-VERSA WHEN NODE *)
(* IS DELETED FROM GO... *)
(*****************************************)

        S1:=(.1..N.)-RR(.NS.); S2:=(.1..N.)-RR(.NR.);
        S1:=S1-(.NI.); S2:=S2-(.NI.);
        FOR I:=1 TO N DO
        IF I IN RR(.NS.) THEN
        FOR J:=1 TO N DO
        IF J IN S1 THEN
        IF C(.I,J.) < MIN THEN MIN:=C(.I,J.);
        FOR I:=1 TO N DO
        IF I IN RR(.NR.) THEN
        FOR J:=1 TO N DO
        IF J IN S2 THEN
        IF C(.I,J.) < MIN THEN MIN:=C(.I,J.);
        FOR I:=1 TO N DO FR(.I.):=(..);
        FOR I:=1 TO N DO
        IF I IN RR(.NS.) THEN
        FOR J:=1 TO N DO
        IF J IN S1 THEN
        IF C(.I,J.) <> INF THEN
        BEGIN C(.I,J.):=C(.I,J.)-MIN; FR(.I.):=FR(.I.)+(.J.)
        END;
        FOR I:=1 TO N DO
        IF I IN RR(.NR.) THEN
        FOR J:=1 TO N DO
        IF J IN S2 THEN
        IF NOT(J IN FR(.I.)) THEN
        IF C(.I,J.) <> INF THEN C(.I,J.):=C(.I,J.)-MIN;
        FORADJ(NN,M); REACH(NI);
        WRITELN; WRITELN;
        WRITE(,   THE COST MATRIX IS REDUCED AGAIN,);
        WRITE(,   NUMBER OF ARCS IN GO =,,M:5)
      END
    UNTIL CON = 1
  END;

(*****************************************)
(* THE SUBGRAPH COMPRISED BY THE ZERO COST ARCS *)
(* IN THE COST MATRIX IS LITERALLY CONNECTED... *)
(*****************************************)

  ROOTNODE(RN); BACKADJ; ALL:=1;
  HAMIL(FAR,BAR,DF,PR,OD,ID,N,M,RN,ALL,FF);
  IF FF = 0 THEN

(*****************)
(* CALL PROCEDURE LIT... *)
(*****************)

  BEGIN LIT(C,MHC,N,INF); MCOST:=0;
    FOR I:=1 TO N DO
    BEGIN I1:=MHC(.I.);
      IF I <> N THEN I2:=MHC(.I+1.) ELSE I2:=MHC(.1.);
      MCOST:=MCOST+D(.I1,I2.)
```

```
    END
  END;
  WRITELN; WRITELN;
  WRITE('   OPTIMUM SOLUTION :');
  WRITELN; WRITELN; I1:=0; I2:=0;
  REPEAT WRITELN('    ');
    I1:=I2+1; I2:=I2+20;
    IF I2 > N THEN I2:=N;
    FOR I:=I1 TO I2 DO WRITE(MHC(.I.):4)
  UNTIL I2 = N;
  WRITELN; WRITELN;
  WRITE('   COST =',MCOST:9)
END;

PROCEDURE YM( VAR C : MATRIX;
              VAR HC :ARRY;
              VAR N,INF : INTEGER );

(*****************************************************************)
(* THIS PROCEDURE FINDS AN OPTIMUM OR A NEAR OPTIMUM SOLUTION    *)
(* TO THE TSP BY RANKING THE EXTREME POINTS ON THE ASSIGNMENT    *)
(* POLYTOPE AND INTRODUCING SOME CUTS SO THAT THE RANKING        *)
(* PROCESS CAN BE PERFORMED IN A MORE EFFICIENT MANNER...        *)
(*****************************************************************)

TYPE ASSIGNMENT = RECORD
                    COLUMN,TOR,ROWMIN,COLMIN,PENALTY : INTEGER
                  END;
     NODES = SET OF 1..ND;
     ARR = ARRAY(.1..ND.) OF INTEGER;

VAR ASS : ARRAY(.1..ND.) OF ASSIGNMENT;
    D,E : MATRIX;
    ID,OD,ASG,MASG,RLBL,CLBL,SORT : ARR;
    TOUR : ARRAY(.1..26.) OF NODES;
    ROW,COL : NODES;
    K,REDC,I,AA,J,FL1,FL2,COST,IC,J,M,Z1,STOP,NN,KC : INTEGER;

PROCEDURE PRT( VAR AS : ARR );

(*****************************************************************)
(* THIS PROCEDURE OUTPUTS THE SOLUTION TO THE AP BY PRINTING     *)
(* EACH ASSIGNMENT SEPARATELY...                                 *)
(*****************************************************************)

VAR I,I1,I2 : INTEGER;

BEGIN WRITELN; WRITELN;
  I1:=0; I2:=0;
  REPEAT WRITELN; WRITE('     ');
    I1:=I2+1; I2:=I2+15;
    IF I2 > N THEN I2:=N;
    FOR I:=I1 TO I2 DO WRITE('(',I:2,',',AS(.I.):2,') ')
  UNTIL I2 = N
END;

PROCEDURE REDUCE;

(*****************************************************************)
(* THIS PROCEDURE REDUCES THE COST MATRIX, THE MINIMUM ELEMENT   *)
(* IN EACH ROW IS FOUND AND SUBTRACTED IT FROM EVERY ELEMENT     *)
(* IN THAT ROW...                                                *)
(*****************************************************************)

VAR I,J,MIN : INTEGER;

BEGIN  (* REDUCE *)
  FOR I:=1 TO N DO
  BEGIN MIN:=INF;
    FOR J:=1 TO N DO
    IF C(.I,J.) < MIN THEN MIN:=C(.I,J.);
    REDC:=REDC+MIN;
    FOR J:=1 TO N DO
    IF C(.I,J.) > MIN THEN
    BEGIN IF C(.I,J.) <> INF THEN C(.I,J.):=C(.I,J.)-MIN
    END ELSE
    IF C(.I,J.) = MIN THEN
    BEGIN OD(.I.):=OD(.I.)+1; ID(.J.):=ID(.J.)+1; C(.I,J.):=0
    END
  END;
  FOR J:=1 TO N DO
  IF ID(.J.) = 0 THEN
```

```
  BEGIN MIN:=INF;
    FOR I:=1 TO N DO
    IF C(.I,J.) < MIN THEN MIN:=C(.I,J.);
    REDC:=REDC+MIN;
    FOR I:=1 TO N DO
    IF C(.I,J.) > MIN THEN
    BEGIN IF C(.I,J.) <> INF THEN C(.I,J.):=C(.I,J.)-MIN
    END ELSE
    IF C(.I,J.) = MIN THEN
    BEGIN ID(.J.):=ID(.J.)+1; OD(.I.):=OD(.I.)+1; C(.I,J.):=0
    END
  END
END;

PROCEDURE ALLOCATE;

(*****************************************************************)
(* THIS PROCEDURE MAKES THE INITIAL ALLOCATIONS AMONG THE        *)
(* ADMISSIBLE CELLS IN THE COST MATRIX WHERE THE ADMISSIBLE      *)
(* CELLS ARE THOSE CELLS WHOSE ENTRIES IN THE REDUCED COST       *)
(* MATRIX ARE ZERO.....                                          *)
(*****************************************************************)

VAR I,J,IC : INTEGER;

BEGIN  (* ALLOCATE *)
  FOR I:=1 TO N DO
  BEGIN
    IF NOT(I IN ROW) THEN
    IF OD(.I.) = 1 THEN
    BEGIN IC:=0; J:=0;
      REPEAT J:=J+1;
        IF NOT(J IN COL) THEN
        IF C(.I,J.) = 0 THEN
        BEGIN K:=K+1;
          ASG(.I.):=J;
          ROW:=ROW+(.I.);
          COL:=COL+(.J.); IC:=1
        END
      UNTIL (IC = 1) OR (J = N)
    END;
    IF NOT(I IN COL) THEN
    IF ID(.I.) = 1 THEN
    BEGIN IC:=0; J:=0;
      REPEAT J:=J+1;
        IF NOT(J IN ROW) THEN
        IF C(.J,I.) = 0 THEN
        BEGIN K:=K+1;
          ASG(.J.):=I;
          ROW:=ROW+(.J.);
          COL:=COL+(.I.); IC:=1
        END
      UNTIL (IC = 1) OR (J = N)
    END
  END;
  IF K < N THEN
  FOR I:=1 TO N DO
  IF NOT(I IN ROW) THEN
  BEGIN IC:=0; J:=0;
    REPEAT J:=J+1;
      IF NOT(J IN COL) THEN
      IF C(.I,J.) = 0 THEN
      BEGIN K:=K+1;
        ASG(.I.):=J;
        ROW:=ROW+(.I.);
        COL:=COL+(.J.); IC:=1
      END
    UNTIL (IC = 1) OR (J = N)
  END
END;

PROCEDURE LBL( VAR C : MATRIX;
               VAR FL1,FL2,AA,L,M : INTEGER );


(*****************************************************************)
(* THIS PROCEDURE LABELS THE ROWS AND COLUMNS AS FOLLOWS :       *)
(* 1- EVERY ROW I THAT HAS NO ALLOCATION IS LABELED AS           *)
(*    RLBL(I)=I                                                  *)
(* 2- IF ROW I IS LABELED AND COLUMN J IS NOT LABELED SO FAR     *)
(*    AND IF CELL (I,J) IS AN ADMISSIBLE CELL AT THIS STAGE ,    *)
(*    THEN COLUMN J IS LABELED AS CLBL(J)=I                      *)
```

```
(* 3- IF COLUMN J IS LABELED AND ROW I IS NOT LABELED SO FAR    *)
(*     AND CELL (I,J) HAS AN ALLOCATION AT THIS STAGE ,THEN ROW *)
(*     I IS LABELED AS RLBL(I)=-J                               *)
(*****************************************************************)

VAR I,J,JC : INTEGER;

BEGIN
  REPEAT FL1:=0; FL2:=0;
    FOR I:=1 TO M DO
    IF RLBL(.I.) <> 0 THEN
    FOR J:=1 TO N DO
    IF CLBL(.J.) = 0 THEN
    IF C(.I,J.) = 0 THEN
    BEGIN JC:=1;
      IF I = M THEN
      IF J IN TOUR(.L.) THEN JC:=0;
      IF JC = 1 THEN
      BEGIN CLBL(.J.):=I; FL1:=1; AA:=AA-1;
        IF NOT(J IN COL) THEN FL2:=1
      END
    END;
    IF FL2 = 0 THEN
    FOR I:=1 TO M DO
    IF RLBL(.I.) = 0 THEN
    BEGIN J:=ASG(.I.);
      IF CLBL(.J.) <> 0 THEN
      BEGIN RLBL(.I.):=-J; FL1:=1; AA:=AA+1
      END
    END
  UNTIL (FL1 = 0) OR (FL2 = 1)
END;

PROCEDURE REALLOCATE;

(*****************************************************************)
(* THIS PROCEDURE CHANGES THE ALLOCATIONS IN ORDER TO OBTAIN     *)
(* THE OPTIMAL SOLUTION TO THE ASSIGNMENT PROBLEM.               *)
(* LET J BE THE COLUMN WHICH DOES NOT HAVE AN ALLOCATION AND     *)
(* HAS BEEN LABELED :                                            *)
(* 1- LET I=CLBL(J). MAKE THE NEW ALLOCATION IN CELL (I,J).      *)
(* 2- LET J=RLBL(I). IF J > 0 THEN STOP ALLOCATION CHANGING.     *)
(*     IF J < 0 THEN REPLACE J=ABS(J) AND REPEAT THESE STEPS     *)
(*****************************************************************)

VAR J,II,JJ,KC : INTEGER;

BEGIN J:=0; II:=0;
  REPEAT J:=J+1;
    IF NOT(J IN COL) THEN
    IF CLBL(.J.) <> 0 THEN II:=CLBL(.J.)
  UNTIL (II > 0) OR (J = N);
  IF II > 0 THEN
  BEGIN JJ:=J; KC:=0;
    REPEAT K:=K+1;
      ROW:=ROW+(.II.); COL:=COL+(.JJ.);
      ASG(.II.):=JJ;
      IF RLBL(.II.) < 0 THEN
      BEGIN JJ:=ABS(RLBL(.II.));
        K:=K-1; COL:=COL-(.JJ.);
        II:=CLBL(.JJ.)
      END ELSE KC:=1
    UNTIL KC = 1
  END
END;

PROCEDURE REDUCE( VAR C : MATRIX;
                  VAR CCC,AA,L,M : INTEGER );

(*****************************************************************)
(* THIS PROCEDURE FINDS THE MINIMUM NON-NEGATIVE ENTRY AMONG     *)
(* THE CELLS IN LABELED ROWS AND UNLABELED COLUMNS OF THE        *)
(* REDUCED MATRIX. IT IS SUBTRACTED FROM THE ENTRIES IN THE      *)
(* CELLS IN LABELED ROWS AND UNLABELED COLUMNS AND ADDED TO      *)
(* CELLS IN UNLABELED ROWS AND LABELED COLUMNS OF THE REDUCED    *)
(* COST MATRIX WHILE ALL THE OTHER ENTRIES ARE BEING REMAINED    *)
(* UNCHANGED....                                                 *)
(*****************************************************************)

VAR MIN,I,J,JC : INTEGER;
```

```
BEGIN MIN:=INF;
  FOR I:=1 TO N DO
  IF RLBL(.I.) <> 0 THEN
  FOR J:=1 TO N DO
  IF CLBL(.J.) = 0 THEN
  BEGIN JC:=1;
    IF I = M THEN
    IF J IN TOUR(.L.) THEN JC:=0;
    IF JC = 1 THEN
    IF C(.I,J.) < MIN THEN
    IF C(.I,J.) > 0 THEN MIN:=C(.I,J.)
  END;
  FOR I:=1 TO N DO
  IF RLBL(.I.) <> 0 THEN
  FOR J:=1 TO N DO
  IF CLBL(.J.) = 0 THEN
  IF C(.I,J.) <> INF THEN C(.I,J.):=C(.I,J.)-MIN;
  FOR I:=1 TO N DO
  IF CLBL(.I.) <> 0 THEN
  FOR J:=1 TO N DO
  IF RLBL(.J.) = 0 THEN
  IF C(.J,I.) <> INF THEN C(.J,I.):=C(.J,I.)+MIN;
  CCC:=CCC+AA*MIN
END;

PROCEDURE CHECK( VAR IC : INTEGER );

(*****************************************************************)
(* THIS PROCEDURE CHECKS IF THE ASSIGNMENT SOLUTION IS A         *)
(* TRAVELING SALESMAN TOUR. IF NOT ALL OF THE SUBTOURS ARE       *)
(* SPECIFIED...                                                  *)
(*****************************************************************)

VAR J,L,M : INTEGER;
    TOURS : NODES;
BEGIN J:=1; L:=0; M:=1; IC:=1; TOURS:=(..);
  REPEAT TOUR(.M.):=(..);
    REPEAT ASS(.J.).COLUMN:=ASG(.J.);
      IF NOT(J IN TOUR(.M.)) THEN
      BEGIN TOUR(.M.):=TOUR(.M.)+(.J.); TOURS:=TOURS+(.J.);
        ASS(.J.).TOUR:=M; L:=L+1;
        J:=ASG(.J.)
      END
    UNTIL J IN TOUR(.M.);
    IF L <> N THEN
    BEGIN IC:=0; J:=1;
      REPEAT J:=J+1
      UNTIL NOT(J IN TOURS);
      M:=M+1
    END
  UNTIL L = N
END;

PROCEDURE CALCULATE(VAR I : INTEGER );

(*****************************************************************)
(* THIS PROCEDURE CALCULATES A NON-NEGATIVE PENALTY ASSOCIATED   *)
(* WITH THE ASSIGNMENT IN ROW I ..                               *)
(*****************************************************************)

VAR J,K,L,M,P : INTEGER;

BEGIN
  WITH ASS(.I.) DO
  BEGIN J:=COLUMN; COLMIN:=INF;
    FOR M:=1 TO N DO
    IF C(.M,J.) < COLMIN THEN
    IF M <> I THEN COLMIN:=C(.M,J.);
    L:=TOUR; ROWMIN:=INF;
    FOR M:=1 TO N DO
    IF NOT(M IN TOUR(.L.)) THEN
    IF C(.I,M.) < ROWMIN THEN ROWMIN:=C(.I,M.);
    PENALTY:=COLMIN+ROWMIN;
    IF PENALTY < 0 THEN
    BEGIN PENALTY:=INF;
      FOR K:=1 TO N DO
      IF K <> I THEN
      FOR M:=1 TO N DO
      IF NOT(M IN TOUR(.L.)) THEN
      BEGIN P:=C(.I,M.)+C(.K,J.);
        IF P < PENALTY THEN
        IF P >= 0 THEN
```

```
        BEGIN PENALTY:=P; ROWMIN:=C(.I,M.); COLMIN:=C(.K,J.)
        END
      END
    END;
    WRITELN; WRITE(,   PENALTY(,,I:2,,,,,COLUMN:2,,)= ,);
    WRITE(PENALTY:6)
  END
END;

PROCEDURE SOLVE( VAR E : MATRIX;
                 VAR I : INTEGER );

(*****************************************************************)
(* THIS PROCEDURE SOLVES THE ASSIGNMENT PROBLEM AFTER A CUT IS   *)
(* INTRODUCED...                                                 *)
(*****************************************************************)

VAR J,M,AA,L,FL1,FL2,CRED,,C,JC : INTEGER;
    SUBLOOP : NODES;

BEGIN
  WITH ASS(.I.) DO
  BEGIN J:=COLUMN; CRED:=PENALTY;
    FOR M:=1 TO N DO
    IF E(.I,M.) <> INF THEN E(.I,M.):=E(.I,M.)-ROWMIN;
    FOR M:=1 TO N DO
    IF E(.M,J.) <> INF THEN E(.M,J.):=E(.M,J.)-COLMIN
  END;
  K:=N-1; ROW:=(.1..N.)-(.I.); COL:=(.1..N.)-(.J.);
  FOR M:=1 TO N DO
  BEGIN RLBL(.M.):=0; CLBL(.M.):=0; ASG(.M.):=ASS(.M.).COLUMN
  END;
  RLBL(.I.):=I; ASG(.I.):=0; AA:=1; L:=ASS(.1.).TOR;
  WHILE K < N DO
  BEGIN LBL(E,FL1,FL2,AA,L,I);
    IF FL2 = 1 THEN REALLOCATE ELSE
    IF FL1 = 0 THEN REDUCE(E,CRED,AA,L,I)
  END;
  WRITELN; WRITELN;
  WRITE(,     INTERMEDIATE SOLUTION :,);
  PRT(ASG); WRITELN; WRITELN;
  WRITE(,     COST = ,,CRED:8);
  IF CRED <= Z1 THEN
  BEGIN JC:=1;
    IF (CRED <= 0) OR (CRED = Z1) THEN
    BEGIN J:=1; L:=0; SUBLOOP:=(..); M:=1;
      REPEAT
        IF NOT(J IN SUBLOOP) THEN
        BEGIN SUBLOOP:=SUBLOOP+(.J.); L:=L+1; J:=ASG(.J.)
        END
      UNTIL J IN SUBLOOP;
      IF L < N THEN JC:=0
    END;
    IF JC =1 THEN
    BEGIN Z1:=CRED; KC:=1;
      FOR M:=1 TO N DO
      BEGIN MASG(.M.):=ASG(.M.);
        FOR J:=1 TO N DO D(.M,J.):=E(.M,J.)
      END
    END
  END
END;

PROCEDURE SORTP;

(*****************************************************************)
(* THIS PROCEDURE SORTS THE PENALTIES IN ASCENDING ORDER...      *)
(*****************************************************************)

VAR I,J,L,M : INTEGER;

BEGIN
  FOR I:=1 TO N DO SORT(.I.):=I;
  REPEAT L:=0;
    FOR I:=1 TO N-1 DO
    BEGIN J:=SORT(.I.); M:=SORT(.I+1.);
      IF ASS(.J.).PENALTY < ASS(.M.).PENALTY THEN
      BEGIN SORT(.I.):=M; SORT(.I+1.):=J; L:=L+1
      END
    END
  UNTIL L = 0
END;
```

```
BEGIN NN:=N DIV 2+1;
  FOR I:=1 TO NN DO TOUR(.I.):=(..);
  K:=0; REDC:=0;
  FOR I:=1 TO N DO
  BEGIN ID(.I.):=0; OD(.I.):=0; ASG(.I.):=0
  END;
  COL:=(..); ROW:=(..); AA:=0;
  REDUCE;
  ALLOCATE;
  IF K < N THEN
  FOR I:=1 TO N DO
  BEGIN CLBL(.I.):=0; RLBL(.I.):=0;
    IF NOT(I IN ROW) THEN
    BEGIN RLBL(.I.):=I; AA:=AA+1
    END
  END;
  WHILE K < N DO
  BEGIN L:=1; M:=0;
    LBL(C,FL1,FL2,AA,L,M);
    IF FL2 = 1 THEN
    BEGIN REALLOCATE;
      IF K < N THEN
      BEGIN AA:=0;
        FOR I:=1 TO N DO
        BEGIN RLBL(.I.):=0; CLBL(.I.):=0;
          IF NOT(I IN ROW) THEN
          BEGIN RLBL(.I.):=I; AA:=AA+1
          END
        END
      END ELSE
      IF FL1 = 0 THEN REDUCE(C,REDC,AA,L,M)
  END;
  COST:=REDC;
  WRITELN; WRITELN;
  WRITE('    SOLUTION TO THE AP :');
  PRT(ASG); WRITELN; WRITELN;
  WRITE('    COST =',COST:8);
  REPEAT CHECK(IC);
    IF IC = 0 THEN
    BEGIN
      FOR I:=1 TO N DO CALCULATE(I);
      SORTP;
      M:=0; Z1:=INF; STOP:=0;
      REPEAT M:=M+1; I:=SORT(.M.);
        FOR L:=1 TO N DO
        FOR J:=1 TO N DO E(.L,J.):=C(.L,J.);
        SOLVE(E,T);
        IF M < N THEN
        BEGIN I:=SORT(.M+1.);
          IF Z1 <= ASS(.I.).PENALTY THEN STOP:=1
        END
      UNTIL (STOP = 1) OR (M = N);
      FOR I:=1 TO N DO
      BEGIN ASG(.I.):=MASG(.I.);
        FOR J:=1 TO N DO C(.I,J.):=D(.I,J.)
      END;
      COST:=COST+Z1;
      WRITELN; WRITELN;
      WRITE('    CURRENT SOLUTION :');
      PRT(MASG); WRITELN; WRITELN;
      WRITE('     COST =',COST:8)
    END
  UNTIL IC = 1;
  WRITELN; WRITELN;
  WRITE('   THE CURRENT SOLUTION IS  OPTIMUM');
  J:=1; HC(.1.):=1;
  FOR I:=2 TO N DO
  BEGIN HC(.I.):=MASG(.J.); J:=MASG(.J.)
  END
END;

PROCEDURE DYNAMIC( VAR C : MATRIX;
                   VAR HC : ARRY;
                   VAR N,INF : INTEGER );

(*****************************************************************)
(* THIS PROCEDURE FINDS THE SOLUTION TO THE TSP BY USING A       *)
(* DYNAMIC PROGRAMMING TYPE APPROACH. FIRST, ALL OF THE CELLS    *)
(* IN THE COST MATRIX ARE SUBTRACTED FROM A LARGE NUMBER SO      *)
(* THAT THE TRIANGULAR INEQUALITY HOLDS. THEN, THE ALGORITHM     *)
```

```
(* TRIES TO FIND THE LRGEST PATH OF CARDINALITY N BETWEEN ANY   *)
(* TWO NODES IN AN ACYCLIC MANNER. ONCE THE LARGEST PATH IS     *)
(* DEFINED, IT IS COMPLETED TO A HAMILTONIAN CIRCUIT. AFTER     *)
(* ALL, THE CIRCUIT IS TRIED TO BE IMPROVED BY ATTEMPTING TO    *)
(* CHANGE THE LOCATION OF EACH NODE SEPARATELY...               *)
(*****************************************************)

 TYPE ARR1 = ARRAY(.1..ND.) OF INTEGER;
      NODES = SET OF 1..ND;
      ARR2 = ARRAY(.1..ND.) OF NODES;

 VAR OD,TO : ARR1;
     BSR : NODES;
     FR,BR : ARR2;
     CMAX,I,J,CMIN,I1,I2 : INTEGER;

 PROCEDURE LP( VAR FR,BR : ARR2;
               VAR BSR : NODES;
               VAR CMIN,CMAX,SR : INTEGER );

(*****************************************************)
(* THIS PROCEDURE FINDS THE LONGEST PATH OF CARDINALITY N IN A *)
(* GRAPH...                                                    *)
(*****************************************************)

 VAR PATH,PATH1 : ARR2;
     S,GS,T : NODES;
     P : MATRIX;
     CR,LK,LK1 : ARR1;
     K,I,MAX,L,J,M,JA,COST : INTEGER;

 BEGIN S:=FR(.SR.); K:=1; GS:=(..);
   FOR I:=1 TO N DO
   BEGIN PATH(.I.):=(.SR.);
     FOR J:=1 TO N DO
     P(.I,J.):=I;
     CR(.I.):=0
   END;
   FOR I:=1 TO N DO
   IF I IN S THEN
   BEGIN LK(.I.):=CMAX-C(.SR,I.);
     GS:=GS+FR(.I.);
     CR(.I.):=1;
     P(.I,1.):=SR
   END ELSE LK(.I.):=0;
   REPEAT
     FOR I:=1 TO N DO
     BEGIN LK1(.I.):=LK(.I.);
       IF I IN GS THEN
       IF I <> SR THEN
       BEGIN T:=BR(.I.)*S; MAX:=0;
         FOR J:=1 TO N DO
         IF J IN T THEN
         IF NOT( I IN PATH(.J.) ) THEN
         BEGIN M:=LK(.J.)+CMAX-C(.J,I.);
           IF M > MAX THEN
           BEGIN MAX:=M; JA:=J
           END
         END;
         IF MAX > LK1(.I.) THEN
         BEGIN LK1(.I.):=MAX;
           PATH1(.I.):=PATH(.JA.)+(.JA.);
           CR(.I.):=K+1;
           P(.I,K+1.):=JA
         END
       END
     END;
     K:=K+1;
     IF (K < N-1) THEN
     BEGIN S:=(..); GS:=(..);
       FOR I:=1 TO N DO
       IF LK1(.I.) <> LK(.I.) THEN
       BEGIN S:=S+(.I.);
         GS:=GS+FR(.I.);
         LK(.I.):=LK1(.I.);
         PATH(.I.):=PATH1(.I.)
       END
     END
   UNTIL (K = N-1);
   FOR I:=1 TO N DO
   IF I IN BSR THEN
   IF (CR(.I.) = N-1) THEN
```

```
    BEGIN COST:=(N-1)*CMAX-LV1(.T.)+C(.I,SR.);
      IF COST < CMIN THEN
      BEGIN CMIN:=COST; HC(.N.):=I; J:=I;
        FOR L:=N-1 DOWNTO 1 DO
        BEGIN HC(.L.):=P(.J,I.); J:=P(.J,L.)
        END
      END
    END
  END;

  BEGIN
    FOR I:=1 TO N DO
    BEGIN FR(.I.):=(..); BR(.I.):=(..); OD(.I.):=0; ID(.I.):=0
    END;
    CMAX:=0;
    FOR I:=1 TO N DO
    FOR J:=1 TO N DO
    IF C(.I,J.) <> INF THEN
    BEGIN FR(.I.):=FR(.I.)+(.J.);
      BR(.J.):=BR(.J.)+(.I.);
      OD(.I.):=OD(.I.)+1;
      ID(.J.):=ID(.J.)+1;
      IF CMAX < C(.I,J.) THEN CMAX:=C(.I,J.)
    END;
    CMAX:=CMAX*2;
    CMIN:=INF;
    BSR:=BR(.1.); BR(.1.):=(..); I:=1;
    LP( FR,BR,BSR,CMIN,CMAX,T );
    WRITELN; WRITELN;
    IF CMIN >= INF THEN
    WRITE(     THE TSP HAS NO SOLUTION,) ELSE
    BEGIN
      WRITE( ,     OPTIMUM SOLUTION :, );
      WRITELN; WRITELN; I1:=0; I2:=0;
      REPEAT WRITELN; WRITE(     ,);
        I1:=I2+1; I2:=I2+20;
        IF I2 > N THEN I2:=N;
        FOR I:=I1 TO I2 DO WRITE(HC(.I.):4)
      UNTIL I2 = N;
      WRITELN; WRITELN;
      WRITE(     COST = ,,CMIN:8)
    END;
    WRITELN; WRITELN;
  END;

  PROCEDURE YH(VAR D : MATRIX;
               VAR HULL : ARRY;
               VAR N,INF,M : INTEGER );

(*****************************************************************)
(* THIS PROCEDURE FINDS THE SOLUTION TO THE TSP BY USING A       *)
(* GEOMETRIC APPROACH. IT IS ASSUMED THAT THE TRIANGULAR         *)
(* INEQUALITY HOLDS. GIVEN A CONVEX HULL THE ALGORITHM TRIES     *)
(* TO FIND THE BEST ALTERNATIVE AMONG THE NODES THAT ARE NOT     *)
(* ON THE CONVEX HULL SO THAT THE TOTAL COST IS MINIMAL AFTER    *)
(* THAT NODE IS INSERTED BETWEEN TWO CONSECUTIVE NODES ON THE    *)
(* CONVEX HULL...                                                *)
(*****************************************************************)

  TYPE EDGES = RECORD
                 ENDNODE,CANDIDATE : INTEGER;
                 HEIGHT : REAL
               END;
  VAR LIST : ARRAY(.1..ND.) OF EDGES;
      TOUR : SET OF 1..ND;
      MAXH,MIN : REAL;
      I,IT,MAX,J,K,JJ,L,I1,I2,COST : INTEGER;

  PROCEDURE MAXIMIZE(VAR HEIGHT : REAL;
                     VAR I,J,L : INTEGER );

(*****************************************************************)
(* THIS PROCEDURE FINDS THE MAXIMUM HEIGHT BELONGING TO ONE OF   *)
(* TRIANGLES FORMED BY ONE OF THE REMAINING NODES THAT ARE NOT   *)
(* NOT ON THE CONVEX HULL AND TWO CONSECUTIVE NODES ON THE       *)
(* CONVEX HULL...                                                *)
(*****************************************************************)

  VAR K : INTEGER;
      U,S,HH : REAL;

  BEGIN HEIGHT:=0.0;
```

```
  FOR K:=1 TO N DO
  IF NOT(K IN TOUR) THEN
  IF D(.I,K.) <> INF THEN
  IF D(.K,J.) <> INF THEN
  BEGIN U:=(D(.I,J.)+D(.I,K.)+D(.K,J.))/2.0;
    S:=SQRT(U*(U-D(.I,J.))*(U-D(.I,K.))*(U-D(.K,J.)));
    HH:=2*S/D(.I,J.);
    IF HEIGHT < HH THEN
    BEGIN HEIGHT:=HH; L:=K
    END
  END
END;

BEGIN TOUR:=(..);
  FOR I:=1 TO N DO
  WITH LIST(.I.) DO
  BEGIN ENDNODE:=0; CANDIDATE:=0; HEIGHT:=0.0
  END;
  FOR I:=1 TO M DO
  BEGIN I1:=HULL(.I.);
    IF I < M THEN I2:=HULL(.I+1.) ELSE I2:=HULL(.1.);
    LIST(.I1.).ENDNODE:=I2;
    TOUR:=TOUR+(.I2.)
  END;
  WRITELN; WRITELN;
  WRITE('     CONVEX HULL :.');
  WRITELN; WRITELN; I1:=0; I2:=0; J:=HULL(.1.);
  REPEAT WRITELN; WRITE('     ');
    I1:=I2+1; I2:=I2+20;
    IF I2 > M THEN I2:=M;
    FOR I:=I1 TO I2 DO
    BEGIN J:=LIST(.J.).ENDNODE; WRITE(J:5)
    END
  UNTIL I2 = M;
  MAX:=0;
  FOR I:=1 TO N DO
  FOR J:=1 TO N DO
  IF D(.I,J.) <> INF THEN
  IF D(.I,J.) > MAX THEN MAX:=D(.I,J.);
  MAX:=2*MAX; COST:=N*MAX;
  FOR I:=1 TO N DO
  FOR J:=1 TO N DO
  IF D(.I,J.) <> INF THEN
  D(.I,J.):=MAX-D(.I,J.);
  FOR I:=1 TO N DO
  IF I IN TOUR THEN
  WITH LIST(.I.) DO
  MAXIMIZE(HEIGHT,I,ENDNODE,CANDIDATE);
  WHILE M < N DO
  BEGIN M:=M+1; MAXH:=0.0;
    FOR L:=1 TO N DO
    IF L IN TOUR THEN
    IF MAXH < LIST(.L.).HEIGHT THEN
    BEGIN MAXH:= LIST(.L.).HEIGHT; T:=L
    END;
    TOUR:=TOUR+(.LIST(.I.).CANDIDATE.);
    IF M <= N THEN
    BEGIN
      WITH LIST(.T.) DO
      BEGIN  L:=ENDNODE;
        ENDNODE:=CANDIDATE; J:=CANDIDATE;
        MAXIMIZE(HEIGHT,I,ENDNODE,CANDIDATE)
      END;
      WITH LIST(.J.) DO
      BEGIN ENDNODE:=L;
        MAXIMIZE(HEIGHT,J,ENDNODE,CANDIDATE)
      END;
      FOR T:=1 TO N DO
      WITH LIST(.T.) DO
      IF CANDIDATE <> 0 THEN
      IF CANDIDATE IN TOUR THEN
      MAXIMIZE(HEIGHT,T,ENDNODE,CANDIDATE)
    END
  END;
  WRITELN; WRITELN;
  WRITE('    OPTIMAL TOUR :.');
  WRITELN; WRITELN; I1:=0; I2:=0; J:=HULL(.1.); L:=J;
  REPEAT WRITELN; WRITE('     ');
    I1:=I2+1; I2:=I2+20;
    IF I2 > N THEN I2:=N;
    FOR I:=I1 TO I2 DO
    BEGIN J:=LIST(.J.).ENDNODE;
```

```
      HULL(.T.):=J; WRITE(J:5);
      COST:=COST+C(.L,J.); L:=J
    END
  UNTIL I2 = N;
  WRITELN; WRITELN;
  WRITE('     COST =',COST:6)
END;

BEGIN
  READLN; READ(N1,INF,TSP,VRP);
  IF VRP = 1 THEN
  BEGIN READLN; READ(NDEM,NDEP);
    DEMAND:=(..); J:=0;
    REPEAT READLN;
      WHILE NOT EOLN DO
      BEGIN J:=J+1;
        IF J <= NDEM THEN
        BEGIN READ(IDD); DEMAND:=DEMAND+(.IDD.)
        END
      END
    UNTIL J >= NDEM;
    DEPOTS:=(..); J:=0; TNV:=0;
    REPEAT READLN;
      WHILE NOT EOLN DO
      BEGIN J:=J+1;
        IF J <= NDEP THEN
        BEGIN READ(IDD,VN(.IDD.));
          DEPOTS:=DEPOTS+(.IDD.);
          TNV:=TNV+VN(.IDD.)
        END
      END
    UNTIL J >= NDEP
  END;
  FOR I:=1 TO N1 DO
  BEGIN J:=0;
    REPEAT READLN;
      WHILE NOT EOLN DO
      BEGIN J:=J+1;
        IF J <= N1 THEN READ(E(.I,J.))
      END
    UNTIL J >= N1
  END;
  PAGE; WRITELN; WRITELN;
  WRITE('     DISTANCE MATRIX :'); WRITELN; WRITELN;
  PRINT(E,N1); WRITELN; WRITELN;
  IF VRP = 1 THEN
  BEGIN WRITE('     NUMBER OF DEMAND POINTS =',NDEM:5);
    WRITELN; WRITELN; WRITE('     DEMAND POINTS :');
    WRITELN; WRITELN; WRITE('     ');
    FOR I:=1 TO N1 DO
    IF I IN DEMAND THEN WRITE(I:3);
    WRITELN; WRITELN;
    WRITE('     NUMBER OF DEPOTS =',NDEM:5);
    WRITELN; WRITELN; WRITE('     DEPOT  # OF VEHICLES');
    WRITELN; WRITELN;
    FOR I:=1 TO N1 DO
    IF I IN DEPOTS THEN
    BEGIN WRITELN; WRITELN;
      WRITE('       ',I:2,'              ',VN(.I.):2)
    END;
    WRITELN; WRITELN;
    WRITE('     TOTAL NUMBER OF VEHICLES =',TNV:5);
    K:=0; N:=TNV+2+NDEM;
    FOR I:=1 TO N DO
    FOR J:=1 TO N DO C(.I,J.):=INF;
    FOR I:=1 TO N1 DO
    IF I IN DEMAND THEN
    BEGIN K:=K+1; NODE(.K.):=I; L:=0;
      FOR J:=1 TO N1 DO
      IF J IN DEMAND THEN
      BEGIN L:=L+1; C(.K,L.):=E(.I,J.)
      END
    END;
    K:=NDEM;
    FOR I:=1 TO N1 DO
    IF I IN DEPOTS THEN
    BEGIN
      FOR M:=1 TO VN(.I.) DO
      BEGIN L:=0; K:=K+1; NODE(.K.):=I;
        FOR J:=1 TO N1 DO
        IF J IN DEMAND THEN
        BEGIN L:=L+1; C(.K,L.):=E(.I,J.)
```

```
        END;
        C(.K,K+1.):=0; K:=K+1; NODE(.K.):=I;
        IF K < N THEN C(.K,K+1.):=0 ELSE C(.K,NDEM+1.):=0
      END
    END;
    K:=NDEM;
    FOR I:=1 TO N1 DO
    IF I IN DEPOTS THEN
    BEGIN
      FOR M:=1 TO VN(.I.) DO
      BEGIN L:=0; K:=K+2;
        FOR J:=1 TO N1 DO
        IF J IN DEMAND THEN
        BEGIN L:=L+1; C(.L,K.):=E(.J,I.)
        END
      END
    END;
    PAGE; WRITELN; WRITELN;
    WRITE(,      TRANSFORMED MATRIX :,);
    WRITELN; WRITELN; PRINT(C,N);
    WRITELN; WRITE(,     NUMBER OF NODES =,,N:5)
  END ELSE
  BEGIN N:=N1;
    FOR I:=1 TO N DO
    FOR J:=1 TO N DO C(.I,J.):=E(.I,J.)
  END;
  CASE TSP OF
    1 : YTSP(C,HC,N,INF);
    2 : YM(C,HC,N,INF);
    3 : DYNAMIC(C,HC,N,INF);
    4 : I:=0
  END;
  IF TSP = 4 THEN
  BEGIN J:=0; READLN; READ(TNV);
    REPEAT READLN;
      WHILE NOT EOLN DO
      BEGIN J:=J+1;
        IF J <= TNV THEN READ(HC(.J.))
      END
    UNTIL J >= TNV;
    YM(C,HC,N,INF,TNV)
  END;
  IF VRP = 1 THEN
  BEGIN M:=0; MM:=0;
    WRITELN; WRITELN; WRITELN;
    WRITE(,     TOURS OF VEHICLES :,);
    WRITELN; WRITELN; I:=0;
    FOR J:=1 TO N1 DO VNU(.J.):=0;
    REPEAT I:=I+1;
      IF I > N THEN I:=1;
      K:=HC(.I.); L1:=NODE(.K.);
      IF L1 IN DEPOTS THEN
      BEGIN I:=I+1;
        IF I > N THEN I:=1;
        K:=HC(.I.); L2:=NODE(.K.);
        IF NOT(L2 IN DEPOTS) THEN
        BEGIN WRITELN; WRITELN; M:=M+1; MM:=MM+2; KM:=2;
          WRITE(,   TOUR ,,M:2,,    ,,L1:3,L2:3);
          REPEAT I:=I+1;
            IF I > N THEN I:=1;
            K:=HC(.I.); L2:=NODE(.K.); KM:=KM+1; MM:=MM+1;
            IF KM <= 40 THEN WRITE(L2:3) ELSE
            BEGIN WRITELN; WRITE(,          ,,L2:3); KM:=0
            END
          UNTIL L2 = L1
        END ELSE
        IF L2 = L1 THEN
        BEGIN VNU(.L2.):=VNU(.L2.)+1; MM:=MM+2
        END ELSE
        BEGIN I:=I-1;
          IF I < 1 THEN I:=N
        END
      END
    UNTIL MM >= N;
    WRITELN; WRITELN;
    FOR I:=1 TO N DO
    IF I IN DEPOTS THEN
    BEGIN VU:=VN(.I.)-VNU(.I.);
      WRITELN; WRITELN;
      WRITE(,    VEHICLES USED IN DEPOT ,,I:2,, = ,,VU:3)
    END
  END
END.
```

# APPENDIX B

```
 DISTANCE MATRIX :

ROW  1     999   65   82   62   62   60   94   33   53   77
ROW  2      65  999   30   53   20    8   34   45   20   22
ROW  3      82   30  999   83   20   38   49   53   29   47
ROW  4      62   53   83  999   69   45   59   70   63   44
ROW  5      62   20   20   69  999   27   51   34    9   42
ROW  6      60    8   38   45   27  999   35   45   24   19
ROW  7      94   34   49   59   51   35  999   79   54   18
ROW  8      33   45   53   70   34   45   79  999   27   64
ROW  9      53   20   29   63    9   24   54   27  999   42
ROW 10      77   22   47   44   42   19   18   64   42  999

 THE COST MATRIX IS REDUCED AGAIN   NUMBER OF ARCS IN GO =   25
 HAMILTONIAN  CIRCUIT :
 4   1   8   9   5   3   2   6  10   7
 COST =     285

 HAMILTONIAN  CIRCUIT :
 4   1   8   9   5   3   2   7  10   6
 COST =     297

 HAMILTONIAN  CIRCUIT :
 4   6  10   7   2   3   5   9   8   1
 COST =     297

 HAMILTONIAN  CIRCUIT :
 4   7  10   6   2   3   5   9   8   1
 COST =     285

NO HAMILTONIAN CIRCUIT CAN BE FOUND

 END OF HAMILTONIAN CIRCUIT SEARCH

 OPTIMUM SOLUTION :

 4   1   8   9   5   3   2   6  10   7

 COST =     285
```

```
 DISTANCE MATRIX :

 ROW  1   999   26   34   80   76   51   42   24   75   17
 ROW  2    26  999   60   96   80   63   47   45   76   22
 ROW  3    34   60  999   70   85   55   57   25   87   45
 ROW  4    80   96   70  999   43   34   53   90   52   74
 ROW  5    76   80   85   43  999   30   15   96    9   63
 ROW  6    51   63   55   34   30  999   19   68   33   41
 ROW  7    42   47   57   53   35   19  999   63   33   28
 ROW  8    24   45   25   90   96   68   63  999   95   40
 ROW  9    75   76   87   52    9   33   73   95  999   60
 ROW 10    17   22   45   74   63   41   28   40   60  999

 SOLUTION TO THE AP :

 ( 1,10) ( 2, 1) ( 3, 8) ( 4, 7) ( 5, 9) ( 6, 4) ( 7, 6) ( 8, 3) ( 9, 5) (10, 2
  COST =   239
  CURRENT SOLUTION :

 ( 1, 2) ( 2, 1) ( 3, 8) ( 4, 6) ( 5, 9) ( 6, 4) ( 7,10) ( 8, 3) ( 9, 5) (10, 7
  COST =   244
  CURRENT SOLUTION :

 ( 1,10) ( 2, 1) ( 3, 8) ( 4, 6) ( 5, 4) ( 6, 7) ( 7, 9) ( 8, 3) ( 9, 5) (10, 2
  COST =   253
  CURRENT SOLUTION :

 ( 1, 7) ( 2, 1) ( 3, 8) ( 4, 6) ( 5, 9) ( 6, 4) ( 7,10) ( 8, 3) ( 9, 5) (10, 2
  COST =   254
  CURRENT SOLUTION :

 ( 1, 8) ( 2,10) ( 3, 1) ( 4, 6) ( 5, 4) ( 6, 7) ( 7, 9) ( 8, 3) ( 9, 5) (10, 2
  COST =   265
  CURRENT SOLUTION :

 ( 1, 8) ( 2, 1) ( 3, 4) ( 4, 6) ( 5, 9) ( 6, 7) ( 7,10) ( 8, 3) ( 9, 5) (10, 2
  COST =   266
  CURRENT SOLUTION :

 ( 1, 8) ( 2, 1) ( 3,10) ( 4, 6) ( 5, 4) ( 6, 7) ( 7, 9) ( 8, 3) ( 9, 5) (10, 2
  COST =   280
  CURRENT SOLUTION :
```

```
( 1, 8) ( 2, 7) ( 3,10) ( 4, 6) ( 5, 9) ( 6, 4) ( 7, 1) ( 8, 3) ( 9, 5) (10, 2)

 COST =      291
 CURRENT SOLUTION :

( 1, 8) ( 2, 7) ( 3, 1) ( 4,10) ( 5, 9) ( 6, 4) ( 7, 6) ( 8, 3) ( 9, 5) (10, 2)
 COST =      297
 CURRENT SOLUTION :

( 1, 8) ( 2,10) ( 3, 4) ( 4, 1) ( 5, 9) ( 6, 7) ( 7, 6) ( 8, 3) ( 9, 5) (10, 2)
 COST =      299
 CURRENT SOLUTION :

( 1, 8) ( 2,10) ( 3, 7) ( 4, 1) ( 5, 9) ( 6, 4) ( 7, 6) ( 8, 3) ( 9, 5) (10, 2)
 COST =      301
 CURRENT SOLUTION :

( 1, 3) ( 2,10) ( 3, 8) ( 4, 1) ( 5, 9) ( 6, 4) ( 7, 6) ( 8, 2) ( 9, 5) (10, 7)
 COST =      305
 CURRENT SOLUTION :

( 1, 3) ( 2,10) ( 3, 8) ( 4, 9) ( 5, 6) ( 6, 4) ( 7, 1) ( 8, 2) ( 9, 5) (10, 7)
 COST =      321
 CURRENT SOLUTION :

( 1, 3) ( 2,10) ( 3, 1) ( 4, 8) ( 5, 9) ( 6, 4) ( 7, 6) ( 8, 2) ( 9, 5) (10, 7)
 COST =      324
 CURRENT SOLUTION :

( 1, 3) ( 2,10) ( 3, 2) ( 4, 8) ( 5, 9) ( 6, 4) ( 7, 6) ( 8, 1) ( 9, 5) (10, 7)
 COST =      329
 CURRENT SOLUTION :

( 1, 8) ( 2, 1) ( 3, 2) ( 4, 9) ( 5,10) ( 6, 4) ( 7, 6) ( 8, 3) ( 9, 5) (10, 7)
 COST =      340
 CURRENT SOLUTION :

( 1, 8) ( 2, 1) ( 3,10) ( 4, 9) ( 5, 2) ( 6, 4) ( 7, 6) ( 8, 3) ( 9, 5) (10, 7)
 COST =      342
THE CURRENT SOLUTION IS OPTIMUM
```

```
OPTIMUM SOLUTION :

  1   2  10   7   6   9   5   4   3   8
COST =      299


CONVEX HULL :

  4   8   1
OPTIMAL TOUR :

  4   6  10   7   2   3   5   9   8   1
COST =   297
```

DISTANCE MATRIX :

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| ROW 1 | 999 | 34 | 68 | 35 | 80 | 42 | 8 | 50 | 60 | 47 |
| ROW 2 | 34 | 999 | 100 | 69 | 101 | 48 | 29 | 84 | 93 | 76 |
| ROW 3 | 68 | 100 | 999 | 34 | 49 | 73 | 72 | 21 | 12 | 28 |
| ROW 4 | 35 | 69 | 34 | 999 | 63 | 55 | 41 | 15 | 25 | 24 |
| ROW 5 | 80 | 101 | 49 | 63 | 999 | 56 | 79 | 61 | 57 | 39 |
| ROW 6 | 42 | 48 | 73 | 55 | 56 | 999 | 37 | 66 | 71 | 45 |
| ROW 7 | 8 | 29 | 72 | 41 | 79 | 37 | 999 | 55 | 64 | 49 |
| ROW 8 | 50 | 84 | 21 | 15 | 61 | 66 | 55 | 999 | 10 | 25 |
| ROW 9 | 60 | 93 | 12 | 25 | 57 | 71 | 64 | 10 | 999 | 27 |
| ROW 10 | 47 | 76 | 28 | 24 | 39 | 45 | 49 | 25 | 27 | 999 |

NUMBER OF DEMAND POINTS = 8

DEMAND POINTS :

1 2 3 5 7 8 9 10

NUMBER OF DEPOTS = 8

| DEPOT | # OF VEHICLES |
|---|---|
| 4 | 1 |
| 6 | 1 |

TOTAL NUMBER OF VEHICLES = 2

TRANSFORMED MATRIX :

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ROW 1 | 999 | 34 | 68 | 80 | 8 | 50 | 60 | 47 | 999 | 35 | 999 | 42 |
| ROW 2 | 34 | 999 | 100 | 101 | 29 | 84 | 93 | 76 | 999 | 69 | 999 | 48 |
| ROW 3 | 68 | 100 | 999 | 49 | 72 | 21 | 12 | 28 | 999 | 34 | 999 | 73 |
| ROW 4 | 80 | 101 | 49 | 999 | 79 | 61 | 57 | 39 | 999 | 63 | 999 | 56 |
| ROW 5 | 8 | 29 | 72 | 79 | 999 | 55 | 64 | 49 | 999 | 41 | 999 | 37 |
| ROW 6 | 50 | 84 | 21 | 61 | 55 | 999 | 10 | 25 | 999 | 15 | 999 | 66 |
| ROW 7 | 60 | 93 | 12 | 57 | 64 | 10 | 999 | 27 | 999 | 25 | 999 | 71 |
| ROW 8 | 47 | 76 | 28 | 39 | 49 | 25 | 27 | 999 | 999 | 24 | 999 | 45 |
| ROW 9 | 35 | 69 | 34 | 63 | 41 | 15 | 25 | 24 | 999 | 0 | 999 | 999 |
| ROW 10 | 999 | 999 | 999 | 999 | 999 | 999 | 999 | 999 | 999 | 999 | 0 | 999 |
| ROW 11 | 42 | 48 | 73 | 56 | 37 | 66 | 71 | 45 | 999 | 999 | 999 | 0 |
| ROW 12 | 999 | 999 | 999 | 999 | 999 | 999 | 999 | 999 | 0 | 999 | 999 | 999 |

NUMBER OF NODES = 12

OPTIMUM SOLUTION :

1 5 10 11 12 9 6 7 3 4 8 2

COST = 284

TOURS OF VEHICLES :

TOUR 1 4 8 9 3 5 10 2 1 7 4

VEHICLES USED IN DEPOT 4 = 1

VEHICLES USED IN DEPOT 6 = 0

## REFERENCES


1. Cook, S.A., "The Complexity of Theorem Proving Procedures", Proc. ACM Symp. Theory of Computing, 3, pp. 151-158, 1971.

2. Karp, R.M., "On the Computational Complexity of Combinatorial Problems", Networks, 5, pp. 45-68, 1975.

3. Russel, R., "An Effective Heuristic for the m-tour TSP with some side conditions", ORSA, 25, pp. 517-524, 1977.

4. Golden, B.L., Magnanti, T.L., and Hguyen, H.Q., "Implementing Vehicle Routing Algorithms", Networks, 7, pp. 113-148, 1977.

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

6. Bellmore, M., and Hong, S., "Transformation of Multi Salesman Problem to the Standard Travelling Salesman Problem", JACM, Vol. 21, No. 3, pp. 500-504, 1974.

7. Orloff, C., "Routing a Fleet of M Vehicles to/from a Central Facility", Networks, Vol. 4, No. 2, pp. 147-162, 1974.

8. Swetska, J., and Huckfeldt, V., "Computational Experience with an M Salesman Travelling Salesman Algorithm", Management Science, Vol. 19, No. 7, pp. 790-799, 1973.

9. Lewis, H.R., and Papadimitriou, C.H., "The Efficiency of Algorithms", Scientific American, 238, No. 1, 1978.

10. Garey, M.R., and Johnson, D.S., Computers and Intractibility A guide to the Theory of np-completeness, W.H. Freeman and Co., San Francisco, 1979.

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

12. Bellmore, M., and Nemhauser, G.L., "The Travelling Salesman Problem: A Survey", ORSA, Vol. 16, pp. 538-558, 1968.

13. Garfinkel, R., and Nemhauser, G.L., Integer Programming, John Wiley, Inc., New York, pp. 354-360, 1972.

14. Held, M., and Karp, R., "The Travelling Salesman Problem and Minimum Spanning Trees", ORSA, Vol. 18, pp. 1138-1162, 1970.

15. Held, M., and Karp, R., "The Travelling Salesman Problem and Minimum Spanning Trees, Part II", Math. Programming, Vol. 1, No. 1, pp. 6-25, 1971.

16. Chavatal, V., "On Hamiltonian Ideals", J. Combinatorial Theory Series B, Vol. 12, No. 2, pp. 163-168, 1972.

17. Christofides, N., Graph Theory, Academic Press Inc., London, 1975.

18. Kauffman, A., Dynamic Programming and Finite Games, Academic Press, New York, 1967.

19. Yau, S.S., "Generation of all Hamiltonian Circuits, Paths and Centres of a Graph and Related Problems", IEEE Trans., CT-14, p. 79, 1967.

20. Danielson, G.H., "On finding the simple paths and circuits in a graph", IEEE Trans., CT-15, p. 294, 1968.

21. Dhawan, V., "Hamiltonian Circuits and Related Problems in Graph Theory", M.Sc. Report, Imperial College, London, 1969.

22. Roberts, S.M., and Flores, B., "Systematic Generation of Hamiltonian Circuits", Comm. of ACM, 9, p. 690, 1966.

23. Selby, G.R., The Use of Topological Methods in Computer Aided Circuit Layout, Ph.D. Thesis, London University, 1970.

24. Christofides, N., "The Shortest Hamiltonian Chain of a Graph", Jl. of SIAM (Appl. Math.), 19, p. 689, 1970.

25. Martello, S., "An Enumerative Algorithm for Finding Hamiltonian Circuits in a Directed Graph", University of Bologna, Italy CR. Categories: 5.32, 8.3, 1980.

26. Barthes, J.P., "Branching Methods in Combinatorial Optimization", in Topics in Combinatorial Optimization Problems, edited by S. Rinaldi, New York, 1975.

27. Eastman, W.L., Linear Programming with Pattern Comstraints, Ph.D. Dissertation, Harvard, 1958.

28. Shapiro, D., Algorithms for the solution of the optimal cost travelling salesman problem, Sc.D. Thesis, Washington University, St. Louis, 1966.

29. Bellmore, M., and Malone, J.C., "Pathology of travelling salesman subtour elimination algorithms", Operations Research, 19, p. 278, 1971.

30. Balas and Christofides, N., " A new penalty method for the travelling salesman problem", Presented at the 9th Math. Prog. Symposium, Budapest, 1976.

31. Christofides, N., Mingozzi, A., Toth, P., and Sandi, C., Combinatorial Optimization, John Wiley and Sons, New York, London, 1979.

32. Hansen, K.H., and Krarup, J., "Improvements of the Held and Karp algorithm the symmetric travelling salesman problem", Mathematical Programming, 4, 87, 1974.

33. Volgerant, T., and Jonker, R., "The symmetric travelling salesman problem and edge exchanges in minimal 1-trees", European Journal of Operational Research, 12, p. 394, 1983.

34. Houck, D.,J., Picard, C., Queyranne, M, and Vemuganti, R.R., "The travelling salesman problem and shortest n-paths", University of Maryland, 1977.

35. Bellman, R., "Dynamic Programming treatment of the travelling salesman problem", J. Assoc. Comp. Mech., 9, 61-3, 1962.

36. Held, M., and Karp, R.M., "A dynamic programming approach to suqencing problems", S.I.A.M., Rev., 10, pp. 196-210.

37. Gonzales, R.H., "Solution to the travelling salesman problem by dynamic programming on a hypercube", Tech. Rep., No. 18, O.R. Center, M.I.T., 1962.

38. Crowder, H., and Padberg, M.W., "Solving large scale symmetric travelling salesman problems to optimality", Management Science, 26, 5, p. 495, 1980.

39. Dantzig, G.B., Fulkersan, D.R., and Johnson, S.M., "Solution of a large scale travelling salesman problem", Operations Research, Vol. 2, pp. 393-410, 1954.

40. Edmonds, J., "Maximum matching and a Polyhedral with 0,1 Vertices", J. Res. National Bureau of Standards, Set. B, Vol. 69, pp. 125-130, 1965.

41. Miliotis, P., "Integer Programming approaches to the travelling salesman problem", Mathematical Programming, 6, p. 367, 1976.

42. Grötschel, M., "An optimal tour through 120 cities in Germany", Report 7770, University of Bonn, 1977.

43. Christofides, N., and Whitlock, C., "An LP-based TSP Algorithm", Imperial College Report OR 78.14, 1978.

44. Rosenkrantz, D.J., Stearns, R.E., and Lewis, P.M., "An analysis of several Heuristics for the travelling salesman problem", SIAM J. Comput., Vol. 6, 3, p. 563, 1977.

45. Norback, J.P. and Love, R.F., "Geometric approaches to solving the travelling salesman problem", Management Science, 23, 11, p. 1208, 1977.

46. Or, İ., "Gezgin satıcı problemi için bulgusal bir algoritma", Yöneylem Araştırması, Bildiriler '77, İstanbul, 1977.

47. Croes, G.A., "A method for solving travelling salesman problems", Operations Research, 6, 791-812, 1958.

48. Reiter, S., and Sherman, G., "Discrete Optimising", SIAM Rev., 13, p. 864, 1965.

49. Lin, S., "Computer solution of the travelling salesman problem", Bell System Tech. J., 44, pp. 2245-2269, 1965.

50. Lin, S., and Kernighan, B.W., "An effective heuristic algorithm for the travelling salesman problem", Operations Res., 21, pp. 498-516, 1973.

51. Minieka, E., Optimization Algorithms for Networks and Graphs, Marcel Dekker Inc., New York and Basel, 1978.

52. Christofides, N., "Worst case analysis of a new heuristic for the travelling salesman problem", Management-Sciences Research Report, No. 388, Carnegie-Mellon University, 1976.

53. Ford, L.R., "Network Flow Theory", Rand Corporation Report, p. 923, 1956.

54. Murty, K.G., "An algorithm for ranking all the assignments in order of increasing cost", Operations Research, 16, pp. 682-687, 1968.

55. Hardgrave, W.W., and Nemhauser, G.L., "On the relation between the travelling salesman problem and the longest path problem", Operations Research, 10, pp. 647-657, 1962.

56. Flood, M.M., "The travelling salesman problem", Operations Research, 4, pp. 61-75, 1956.

57. Gonzales, R.H., "Solution to the travelling salesman problem by dynamic programming on the hypercube", Tech. Rep. No. 18, O.R. Center, M.I.T.

58. Bodin, L., and Golden, B., "Classification in Vehicle Routing and Scheduling", Networks, Vol. 11, pp. 97-108, 1981.

59. Gillett, B., and Miller, L., "A heuristic algorithm for the vehicle dispatch problem", Operations Research, 22, pp. 340-349, 1974.

60. Gillet, B., and Johnson, T., "Multi-terminal vehicle dispatch algorithm", Omega, 4, pp. 711-718, 1976.

61. Karp, R., "Probabilistic analysis of partitioning algorithms for the travelling salesman problem in the plane", Math. Oper. Res., 2, pp. 209-224, 1977.

62. Newton, R., and Thomas, W., "Bus routing in a Multi-school system", Comput. Oper. Res., 1, pp. 213-222, 1974.

63. Bodin, L., and Berman, L., "Routing and scheduling of school busses by computer", Transp. Sci., 13, pp. 113-129, 1979.

64. Bodin, L., and Kursh, S., "A computer assisted system for the routing and scheduling of street sweepers", Oper. Res., 26, pp. 525-537, 1978.

65. Clarke, G., and Wright, J., "Scheduling of vehicles from a central depot to a number of delivery points", Oper. Res., 12, pp. 568-581, 1964.

66. Golden, B., Bodin, L., Doyle, T., and Steward, W., "Approximate travelling salesman algorithms", Oper. Res., 28, pp. 694-711, 1980.

67. Christofides, N., Eilon, S., "An algorithm for the vehicle dispatching problem", Oper. Res. Q., 20, pp. 309-318, 1969.

68. Bodin, L., and Sexton, T., "The subscriber Dial-a-Ride problem", Final Report, U.S. Department of Transportation, Urban Mass Transportation Administration, Washington, D.C., 1979.

69. Fisher, M.L., and Jaikumar, R., "A generalized assignment heuristic for vehicle routing", Networks, Vol. 11, pp. 109-124, 1981.

70. Krolak, P., Felts, W., and Marble, G., "A man machine approach toward solving the travelling salesman problem", Comm. ACM, 14, pp. 327-334, 1971.

71. Krolak, P., Felts, W., and Nelson, J., "A man machine approach toward solving the generalized truck dispatching problem", Transp. Sci., 6, pp. 149-169, 1972.