A POLYHEDRAL APPROACH TO DELIVERY MAN PROBLEM

A THESIS SUBMITTED TO
THE DEPARTMENT OF INDUSTRIAL ENGINEERING
AND THE INSTITUTE OF ENGINEERING
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

by
Pınar Keskinocak
1992

A POLYHEDRAL APPROACH TO DELIVERY MAN PROBLEM

A THESIS SUBMITTED TO
THE DEPARTMENT OF INDUSTRIAL ENGINEERING
AND THE INSTITUTE OF ENGINEERING
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

by
Pınar Keskinocak
1992

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc.Prof. Mustafa Akgül (Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc.Prof. M.Cemal Dinçer

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc.Prof. Osman Oğuz

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc.Prof. Barbaros Tansel

Approved for the Institute of Engineering and Sciences:

Prof. Mehmet Baray

Director of Institute of Engineering and Sciences

ABSTRACT

A POLYHEDRAL APPROACH TO DELIVERY MAN PROBLEM

Pınar Keskinocak
M.S. in Industrial Engineering
Supervisor: Assoc. Prof. Mustafa Akgül
1992

In this thesis we discuss some polyhedral approaches to the Delivery Man Problem(DMP),which is a special case of the Traveling Salesman Problem(TSP). First, we look at two formulations of the problem and describe a combinatorial solution procedure for the linear programming relaxation.Then we give some valid inequalities and discuss a Lagrangean Relaxation procedure and a cutting plane procedure. Finally, we propose heuristics for tree graphs and general graphs and give computational results.

Keywords : Delivery Man Problem, Polyhedral Approach, Cutting Plane.

ÖZET

DELIVERY MAN PROBLEMİNE POLİHEDRAL YAKLAŞIMLAR

Pınar Keskinocak
Endüstri Mühendisliği Bölümü
Yüksek Lisans
Tez Yöneticisi: Doçent Mustafa Akgül
1992

Bu tezde, Delivery Man Problemi'ne polihedral yaklaşımlar tartışılmaktadır. Öncelikle problemin iki değişik formülasyonu verilmiş ve doğrusal programlama gevşetmesi için kombinatoryal bir çözüm yöntemi geliştirilmiştir. Daha sonra bazı geçerli eşitsizlikler belirtilerek, Lagrangean gevşetmesi ve kesen düzlemler prosedürleri tartışılmıştır. Son olarak genel graflar ve ağaçlar için sezgisel yordamlar önerilmiştir.

Anahtar kelimeler: Delivery Man Problemi, Polihedral Yaklaşımlar, Kesen Düzlemler.

To my family,

ACKNOWLEDGEMENT

I wish to express deep gratitude to my supervisor Assoc.Prof. Mustafa Akgül for his supervision, encouragement and patience throughout the development of this study. I am grateful to Assoc.Prof. Cemal Dinçer, Assoc.Prof. Osman Oğuz and Assoc.Prof. Barbaros Tansel for their valuable comments on the thesis.

I wish to thank to my family for providing morale support and encouragement during the preperation of this thesis.

I also wish to express my appreciation to Dr. Cemal Akyel and Ceyda Oğuz for their comments and helps, and I want to thank to all friends and to Assist.Prof. Selim Aktürk, who provided morale support and encouragement.

Contents

Chapter 1

INTRODUCTION

The Traveling Salesman Problem(TSP) is one of the famous combinatorial optimization problems. Its importance comes from the fact that it is a typical example of other combinatorial optimization problems and has a wide area of applications. Many problems (e.g. job sequencing, computer wiring, cutting wallpaper, collection and delivery problems, circuit board drilling, order picking in a warehouse) can be formulated as TSP. If a good algorithm can be found to solve the TSP, those related problems can also be solved efficiently. So, TSP is a testing stage for new theory and algorithms in combinatorial optimization.

In TSP we are given a (directed) graph G=(V,E), where V = {1,..,n} is the node set and E = {(i,j) : there exists an arc from node i to node j} is the arc set. We may assume that nodes represent cities and arcs represent (one way) roads between the cities. Let c_{ij} denote the cost of the arc (i,j) or the travel time from city i to city j. The aim is to find a tour starting from a city (say city 1), visit all other cities exactly once and then return back to city 1, such that the total length of the tour is minimum. In other words, the problem is to find a minimum length Hamiltonian tour in the (directed) graph G.

Several different formulations are introduced for the TSP. The classical formulation is given by Dantzig, Fulkerson and Johnson(1954), which consists of the

assignment problem constraints, plus integrality and subtour elimination constraints. The number of subtour elimination constraints in the classical formulation is exponential. Miller, Tucker and Zemlin(1960) proposed an extended formulation based on the assignment model but using a polynomial number of subtour elimination constraints. Finally, Gavish and Graves(1978) proposed a formulation using $O(n^2)$ binary variables, $O(n^2)$ continuous variables and n^2+3n constraints. There are also several single commodity, two-commodity and multi-commodity flow formulations of TSP.

TSP can be formulated as a 'decision' problem by adding a bound B to the input data. In the 'decision' version of TSP the question is :"Is there a Hamiltonian tour with length less than or equal to B ?". There are two major classes of decision problems : P and NP. The class P consists of those decision problems, for which a polynomial time algorithm exists, where the number of elementary operations (addition, multiplication, comparison etc.) is bounded by a polynomial in the size of the problem. For the decision problems in the class NP, there exists only nondeterministic polynomial algorithms. A nondeterministic algorithm has two stages : a guessing stage and a checking stage. There are usually a large number of guesses, but it can be verified in polynomial time, whether the answer of the decision question for a particular guess is YES. TSP is a problem in NP class. Notice that polynomial verifiability does not imply polynomial solvability.

A problem is said to be NP-complete, if it is in the class NP and if every problem in NP is polynomially transformable to it. TSP is shown to be NP-complete, so it is a 'difficult' problem. TSP remains NP-complete, even if $c_{ij} \in \{1,2\}$ for all (i,j) $\in$ E.

In this thesis we examined the Delivery Man Problem(DMP), which is a variant of TSP. We first give the definition of the DMP and look at some special cases. In section 2, we look at the previous work done about DMP and examine Minieka's algorithm(1989) in more detail. In section 3.1 we give two shortest path formulations of the DMP : the natural shortest path formulation and the extended shortest path formulation. In section 3.2 we describe some valid inequalities for the DMP, which are based on a relaxation of the natural formulation. In section 3.3 we discuss a Lagrangean Relaxation approach, which again uses a relaxation

of the shortest path formulation, but was not found to be very effective in our limited experiment. Section 3.4 consists of a cutting plane approach based on the valid inequalities discussed in section 3.2. In section 3.5 we discuss some heuristics for general graphs and for tree graphs. Finally we give some computational results about our cutting plane procedure.

1.1 Definition of the Delivery Man Problem

There is a wide range of problems, which are generalizations of TSP or relaxations of it. The assignment problem, the quadratic assignment problem, the longest path and shortest path problems, the minimum spanning tree problem, the 2-matching problem are some of the relaxations of TSP.

The Time Dependent Traveling Salesman Problem(TDTSP) is a more general case of the TSP. The feasible solutions of TSP and TDTSP are the same, the difference is in their objective functions. In TDTSP, the cost of the arc (i,j) depends not only on the corresponding nodes i and j, but also on the position of that arc in the tour. In TDTSP costs are represented by c_{ij}^t, which is the cost of going from city i to city j at t-*th* position (i.e. city i is the (t-1)-*st* visited city and j is the t-*th* visited city during the tour). In that respect, TSP is a special case of TDTSP, where costs do not change with the position of the arc on the tour. In TSP, the starting point of the tour does not change the optimum solution, but in TDTSP it does. In TDTSP, the starting point of the tour or the 'root' must be specified at the beginning, since the arc cots are sequence dependent. Usually, node 1 is chosen as the root and TDTSP is called a 'rooted' problem. A typical application of the TDTSP is the job sequencing on a single machine, where the setup times (or costs) vary with the processing sequence of the jobs.

The Delivery Man Problem(DMP) is a special case of the TDTSP. The aim is again to find a Hamiltonian tour in the graph, but the objective is to minimize the sum of the arrival times at the nodes. DMP is also a 'rooted' problem, since the arrival times are sequence dependent. Let c_{ij} denote the cost or the travel

time of the arc (i,j) as in TSP. Let A_j be the arrival time at vertex j. If we go from city i to city j at t-*th* position, its contribution to the objective function is c_{ij}^t, where

$$c_{ij}^t = A_i + c_{ij}$$

Let $i_0, i_1, ..., i_j, i_{j+1}, ..., i_n$ be a Hamiltonian tour on the graph, where $i_0 = i_n = 1$ and i_j is the j^{th} vertex on the tour. Then

$$A_{i_1} = c_{i_0 i_1}$$

$$A_{i_2} = A_{i_1} + c_{i_1 i_2}$$

$$A_{i_j} = A_{i_{j-1}} + c_{i_{j-1} i_j}$$

$$A_{i_n} = A_{i_{n-1}} + c_{i_{n-1} i_n}$$

which implies

$$A_{i_j} = \sum_{k=0}^{j-1} c_{i_k i_{k+1}} \quad , \quad j = 1, ...n-1$$

$$A_1 = c_{i_{n-1} 1} + \sum_{k=0}^{n-2} c_{i_k i_{k+1}}$$

and the objective function of the DMP is

$$z = \min \sum_{j=1}^{n} A_{i_j}$$

DMP can also be perceived as a sequencing problem. Usually, the processing times of the jobs are constant, i.e. do not depend on the preceding or following jobs. But the setup times may be sequence dependent. Let us define the operating time of a job as the sum of the setup and processing times. Then operating times will also be sequence dependent. We can interpret the nodes on the graph as the jobs and the arc cost as the operating times. Cost of (i,j) may be different than cost of (k,j), which means that, we must take c_{ij} as the operating time of job j, if we process it after job i, and we must take c_{kj}, if we process it after job k. If the jobs are processed on a single machine, then the objective of the DMP is to minimize the sum of the completion times of all the jobs.

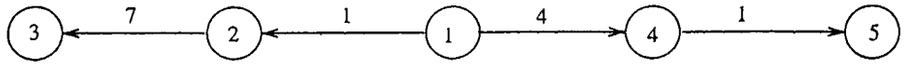
1.2 Special Cases

In this section we look at some easily solved cases of TSP and DMP. Due to the differences between the objective functions of TSP and DMP, 'easy' cases for TSP are not always 'easy' for DMP.

Tree Graphs :

If the graph under consideration is a tree, the TSP is solved by a depth first route, that is any route, in which the salesman arriving at a vertex arbitrarily travels along any edge that he has not yet traversed. If no such edge is available, he traverses again the unique edge in the direction of vertex 1, which is the root of the tree. Every depth first route gives an optimal solution to the TSP. But if we consider the DMP on a tree, there are different total times for different depth first routes, and besides this, a depth first route in a tree may not be optimal for the DMP.

EXAMPLE



There are two depth first routes in this example :

I : 1-2-3-4-5-1 : $z = 1 + 8 + 20 + 21 + 26 = 76$

II: 1-4-5-2-3-1 : $z = 4 + 5 + 11 + 18 + 26 = 64$

But the route, which minimizes z is neither route I nor route II. It is

1-2-4-5-3-1 : $z = 1 + 6 + 7 + 20 + 28 = 62$

So, the TDTSP is not easy to solve even on tree graphs.

Two cases, where the DMP can easily be solved are the star graph and the

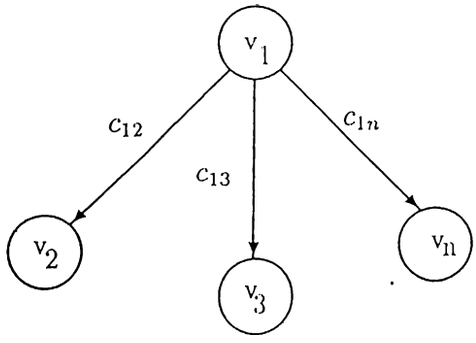
graph with unit edge weights.

Graphs With Unit Edge Weights :

In this case any route is optimal for the DMP. If the graph under consideration is a tree, then any depth first route is optimal.

Star Graphs :

Let $i_1, i_2, ..., i_n$ be any permutation of the indices 1,2,...,n, where $i_n = 1$, since the salesman comes back to node 1 after visiting all other nodes.



The arrival time of the first visited vertex will be

$$A_{i_1} = c_{1i_1}$$

and the arrival time of the k^{th} visited vertex will be

$$A_{i_k} = 2c_{1i_1} + 2c_{1i_2} + ... + c_{1i_k},\ \text{k¿=2}$$

Then the sum of the arrival times is equal to

$$\begin{aligned} z &= A_{i_1} + \textstyle\sum_{k=2}^{n} A_{i_k} \\ &= A_{i_1} + \textstyle\sum_{k=2}^{n} 2c_{1i_1} + 2c_{1i_2} + ... + c_{1i_k} \end{aligned}$$

If we expand the double summation, we obtain

$k = 1 \qquad A_{i_1} = c_{1i_1}$

$k = 2 \qquad A_{i_2} = 2c_{1i_1} + c_{1i_2}$

$k = n-2 \quad A_{i_{n-2}} = 2c_{1i_1} + 2c_{1i_2} + \ldots + 2c_{1i_{n-3}} + c_{1i_{n-2}}$

$k = n-1 \quad A_{i_{n-1}} = 2c_{1i_1} + 2c_{1i_2} + \ldots + 2c_{1i_{n-2}} + c_{1i_{n-1}}$

$k = n \qquad A_{i_n} = A_1 = A_{i_{n-1}} + c_{i_{n-1}1}$

$$z = 2nc_{1i_1} + 2(n-1)c_{1i2} + \ldots + 2c_{1in-1}$$

In order to minimize z, we should choose

$c_{1i1} \leq c_{1i2} \leq \ldots \leq c_{1in-1}$

That means, the vertices should be visited in increasing order of their proximity to the root, in order to minimize the sum of the arrival times.

Chapter 2

LITERATURE REVIEW

The first use of the term 'traveling salesman problem' goes back to 1930's, but the first paper about the TSP is published in 1954 by Dantzig, Fulkerson and Johnson, where they gave a formulation of the problem and proposed for the first time a 'cutting plane' approach for TSP.

The earliest formulation of the TDTSP is due to Fox(1973). Fox gives five linear integer programming formulations and one flow-with-gains formulation for the TDTSP. He reports that his branch and bound algorithm was unable to solve a 10-job problem in 12 minutes and Picard and Queyranne(1978) reported that none of the formulations of Fox were found to lead to a tractable solution scheme.

Sahni and Gonzalez(1976) have shown that the TDTSP is an NP-complete problem.

Picard and Queyranne(1978) proposed an exact algorithm to solve the TDTSP. They state it as a scheduling problem, in which n jobs have to be processed at minimum cost on a single machine. In their approach, the setup costs associated with each job depend both on its position in the sequence and on the job, which precedes it. They propose a branch and bound algorithm, where the branch and bound enumeration is applied after finding shortest paths and doing subgradient

optimization. They give computational results for the weighted tardiness problem for 15 and 20 jobs.

Lucena(1989) proposes a branch and bound algorithm based on a special lower bounding scheme. This scheme involves splitting lower bounds into a number of components and optimizing each of these components. Lucena reports computational results for graphs up to 30 nodes, where the graphs are assumed to be on the Euclidean plane and the coordinates of the nodes were drawn from the uniform distribution in the range [0,100] and [0,1000].

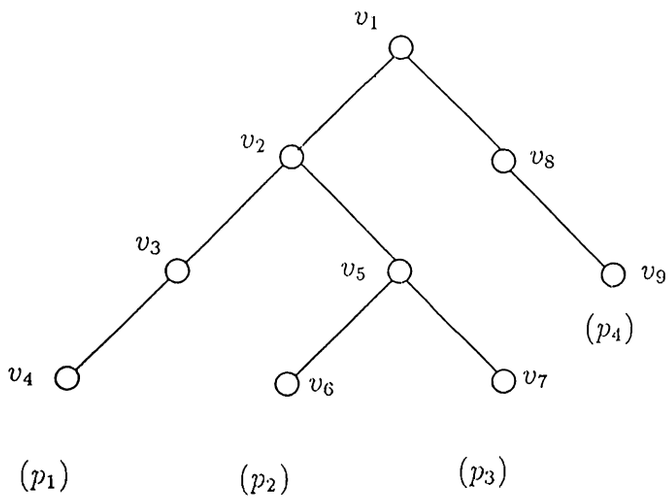
Simchi-Levi and Berman(1991) interpret the TDTSP as a sequencing problem. They try to minimize the total flow time of jobs where flow times are the length of time jobs spend in the system. They propose a branch and bound algorithm to find the optimal job sequence on a single machine, where the jobs have sequence dependent setup times. In addition to finding the optimal sequence, they show how to find the optimal home location for the server.They report that their algorithm can handle efficiently problems with up to 20 nodes.

Minieka(1989) has proposed a pseudo polynomial time solution for the TDTSP, where the graph under consideration is a tree. He suggests to construct a new network, in which each vertex will represent a partial route on the tree. In the following section we want to look at Minieka's algorithm in more detail.

Minieka's Algorithm

Let E be the edge set of the tree. Then, the vertices of the tree can be numbered as 1,2,...,n, such that if (i,j) $\in E$ and i is closer to vertex 1 (root) than j, then $i < j$. Let n denote the number of vertices in T. A vertex is called 'leaf', if there is only one arc incident to that vertex in the tree. Let $p_1, ..., p_k$ denote the leaf vertices in T. Partition the vertices into sets $V_1, V_2, ..., V_k$ as follows : let V_1 consist of all vertices on the unique path from vertex 1 (root) to p_1. For i = 2,..,k let V_i be the set of all vertices on the unique path from root to p_i that are not included in V_1 through V_{i-1}.

Consider a (k+1)-tuple INDEX, in which the first component can take any integer value from 1 to n. This component denotes the subscript of the vertex, that is the current position of the salesman. The (i+1)-*st* component denotes how far the salesman has penetrated into set V_i, with zero indicating that he has not reached any vertex in V_i. The salesman can travel only from one position to another with the same or higher values in all components of INDEX, except possibly the first.



$V_1 = \{v_1, v_2, v_3, v_4\}$

$V_2 = \{v_5, v_6\}$

$V_3 = \{v_7\}$

$V_8 = \{v_8, v_9\}$

Minieka suggests that the TDTSP can be reformulated as a shortest path problem from the source to the sink in a directed acyclic network, in which the vertices correspond to the (k+1)-tuple INDEXes and the arc costs equal the sum of the realized delivery times during the corresponding journey through the tree. The source corresponds to the (1,1,0,0,...) and the sink corresponds to the (k+1)-tuple $1, f(p_1), f(p_2), ..., f(p_k))$, where $f(p_i)$ indicates the number assigned to vertex p_i.

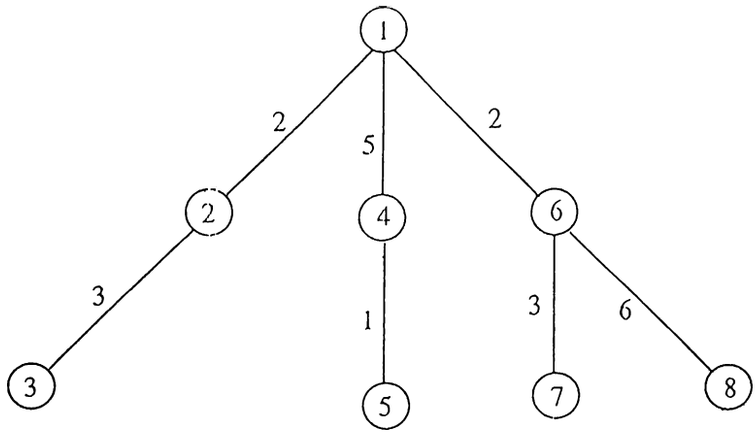
But there are two drawbacks of this formulation :

i) INDEX does not represent the route history of the delivery man, i.e. there may be more than one route which should be represented by the same INDEX.

ii) The arc costs of the new created graph are dynamic, in the sense that each arc cost depends on the previously traversed arc. In this case, any shortest path

algorithm may end up with suboptimal results.

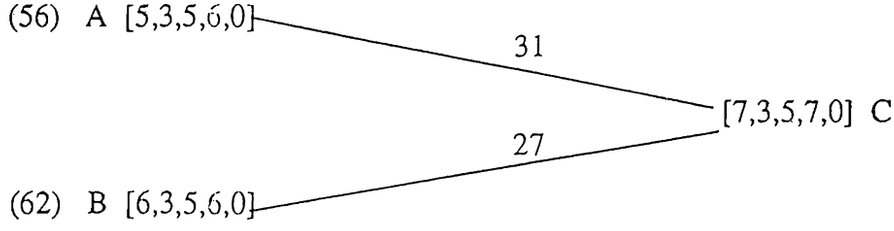
EXAMPLE



Let us partition the nodes into sets a follows :

$V_1=\{1,2,3\}$, $V_2=\{4,5\}$, $V_3=\{6,7\}$ and $V_4=\{8\}$.

Consider the paths 1-2-4-6 and 1-4-2-6. The INDEX corresponding to both of these paths is [6,2,4,6,0]. The first component of the INDEX is 6, since the current position of the delivery man is node 6. The second component of the index correspond to set V_1 and its value is 2, since node 2 is the last visited vertex in set V_1. The third and fourth components of the INDEX are defined similarly. The last component of the INDEX is zero, since the delivery man has not visited any vertex in V_4 yet.But this index does not really tell us which route the delivery man has taken, and this is an example for the first drawback of the algorithm.



It is easy to see that two parents of node [7,3,5,7,0] are [5,3,5,6,0] and [6,3,5,6,0].

If we choose path A-C, i.e. path 1-6-2-3-4-5-7, sum of the arrival times at node C will be 2+6+9+19+20+31=87. If we choose path B-C, i.e. path 1-2-3-4-5-6-7, sum of the arrival times at node C will be 2+5+15+16+24+27=89. Therefore path A-C is chosen. But this is actually not the optimum path. The choice of path A-C caused the arrival time at vertex 7 to be 31, whereas it would be only 27, if we had chosen path B-C. Since the arrival time of a vertex is dependent on the arrival times of the previous vertices, the arrival times of vertices 1 and 8 are also higher by path A-C than by path B-C. At the end, choosing the path A-C results in a z value of 175, whereas choosing B-C would give a z value of 169. This example shows that because of the dynamic nature of the arc costs, the algorithm may end up with suboptimal results.

Chapter 3

NEW APPROACHES

3.1 Formulation of the Problem

We considered two formulations for the DMP, which are based on the shortest path problem. We decided to use the relaxation of the natural shortest path formulation for our cutting plane procedure and its extended version to find the optimal solution.

3.1.1 Natural Shortest Path Formulation

Without loss of generality we may assume that the graph under consideration is a complete directed graph G. We split node 1 into two nodes, 1 and n+1, to obtain a new graph G'. All arcs of the form (1,j), j=2,...,n in the original graph are represented in the same form in the new graph. The arcs (j,1), j=2,...,n in the original graph are represented as (j,n+1), j=2,...,n in the new graph G'. Matrix A (m by n) represents the node arc incidence matrix, where each row corresponds to a node and each column corresponds to an arc. Let a_{ik} denote the element in the i^{th} row and k^{th} column of A. Then

$$a_{ij} = \begin{cases} -1 & , \text{ if node i is the tail of the arc k} \\ +1 & , \text{ if node i is the head of the arc k} \\ 0 & , \text{ otherwise} \end{cases}$$

$x_{i,j}$ = amount of flow from node i to node j

E = arc set

A = node-arc incidence matrix

$$\min \sum_{(i,j)\in E} c_{i,j}x_{i,j}$$

$$\text{s.t. } Ax = b = \begin{cases} -n & , \ i = 1 \\ 1 & , \ i = 2, ..., n+1 \end{cases} \qquad (1.1)$$

$$\sum_{(i,j)\in E} x_{i,j} = \begin{pmatrix} n+1 \\ 2 \end{pmatrix} \qquad (1.2)$$

x defines a Hamiltonian path (1.3)

$x_{i,j} \geq 0 \qquad \forall$ (i,j) $\in$ E (1.4)

x integer (1.5)

Constraints (1.1) provide that the solution is a spanning tree, i.e. connected. Due to constraint (1.2), total flow in the solution must be equal to the total flow on a Hamiltonian path. But constraints (1.1) and (1.2) alone do not prevent the subtours. The solution of the problem subject to (1.1) and (1.2) will most likely be a tree with one cycle only, where (1.2) is satisfied by passing enough amount of flow (usually not integral) along the cycle. Therefore, to obtain a Hamiltonian path as a solution, we need also constraints (1.3) and (1.4), so that the support of the solution will be cycle free.

3.1.2 Extended Shortest Path Formulation

x_{ij} = amount of flow from node i to node j

$$y_{ij} = \begin{cases} 1 & , \text{ if } x_{ij} > 0 \\ 0 & , \text{ otherwise} \end{cases}$$

E = arc set

A = node-arc incidence matrix

$$\min \sum_{(i,j)\in E} c_{i,j} x_{i,j}$$

$$\text{s.t. } Ax = b = \begin{cases} -n & , \ i = 1 \\ 1 & , \ i = 2, ..., n+1 \end{cases} \tag{2.1}$$

$$\sum_{(i,j)\in E} x_{i,j} = \begin{pmatrix} n+1 \\ 2 \end{pmatrix} \tag{2.2}$$

$$x_{i,j} \leq n y_{i,j} \qquad \forall (i,j) \in E \tag{2.3}$$

$$\sum_{j} y_{i,j} = 1 \qquad i = 1, ..., n \tag{2.4}$$

$$\sum_{i} y_{i,j} = 1 \qquad j = 1, ..., n \tag{2.4'}$$

$$x_{i,j} \geq 0 \qquad \forall (i,j) \in E \tag{2.5}$$

$$x_{i,j} \quad \text{integer} \tag{2.6}$$

$$y_{i,j} \in \{0, 1\} \tag{2.7}$$

3.2 Combinatorial Solution Procedure for LP Relaxation

Let us consider a linear programming relaxation(LPR) of the DMP, which is obtained using the shortest path formulation in section 3.1.1.

$x_{i,j}$ = amount of flow from node i to node j

E = arc set

$$\min \sum_{(i,j)\in E} c_{i,j}x_{i,j}$$

$$\text{(LPR)} \qquad \text{s.t. } Ax = b = \begin{cases} -n & , \ i = 1 \\ 1 & , \ i = 2, ..., n+1 \end{cases} \tag{3.1}$$

$$\sum_{(i,j)\in E} x_{i,j} = \begin{pmatrix} n+1 \\ 2 \end{pmatrix} \tag{3.2}$$

$$x_{i,j} \geq 0 \qquad \forall \text{ (i,j)} \in \text{E} \tag{3.3}$$

This formulation can give two types of solutions. One of them is a Hamiltonian path from node 1 to node n+1, which is the optimal solution of the original problem. This is the fortunate case but its occurence is quite unlikely. Most of the time,a second type of solution occurs, which is a tree with one cycle only. The cycle comes from the last constraint. If we disregard that constraint, we obtain a shortest path tree.

The number of variables in the LPR is $(n-1)^2$ and the number of constraints is n+2, where n is the number for nodes in G.

Here we propose a combinatorial procedure to solve LPR. The procedure uses the idea of Dual Simplex method. We start with an initial solution, which is

most probably infeasible due to the flow constraint (3.2). At each step, we try to decrease the infeasibility by entering a new arc and increasing total flow,and at the same time we try to keep the increase in the objective function value at a minimum level.

Let π be the vector of dual variables. Let $\bar{c}$ be the vector of reduced costs, where $\bar{c}_{ij} = c_{ij} - \pi_i + \pi_j - \lambda$.

STEP 0: Find the shortest path tree on graph G', call it T_0. Set i=0, $\pi_0 = \pi$, $\lambda = 0$.

STEP 1: If $\sum_{(i,j)\in E} x_{i,j} < \begin{pmatrix} n+1 \\ 2 \end{pmatrix}$ then go to STEP 2, else STOP.

STEP 2: Let C_{ij}^{+} be the number of arcs in the cycle created by adding arc (i,j) to the tree T_i, which have the same direction as the arc (i,j). Let C_{ij}^{-} be the number of arcs in the cycle, which have reverse direction as the arc (i,j). Find $\frac{\bar{c}_{kl}}{C_{kl}^{+} - C_{kl}^{-}} = \min \frac{\bar{c}_{ij}}{C_{ij}^{+} - C_{ij}^{-}}$, where $C_{kl}^{+} - C_{kl}^{-} > 0$. Let (k,l) be the entering arc.

STEP3 : Set $\Delta\lambda = \frac{\bar{c}_{kl}}{C_{kl}^{+} - C_{kl}^{-}}$

STEP 4 : Update π's in the following way : Find the levels of the nodes in tree T_i, such that the level of the root is 1, the level of a node adjacent to the root is 2, etc. Keep π_l constant.Set $\pi_j = \pi_j + \Delta\lambda(level(j) - level(l))$.

STEP 5 : Set $\bar{c}_{ij} = c_{ij} - \pi_i + \pi_j - \lambda$. Set $\lambda = \lambda + \Delta\lambda$. Set i=i+1.

STEP 6 : Add the arc (k,l) to the tree T_{i-1} and delete the arc (j,l) to obtain the new tree T_i.

STEP 7 : Update flows and go to STEP 1.

Updating the reduced costs can be done without explicitly updating the dual variables. We can simply set $\bar{c}_{ij} = \bar{c}_{ij} + \Delta\lambda(level(j) - level(i) - 1)$

EXAMPLE

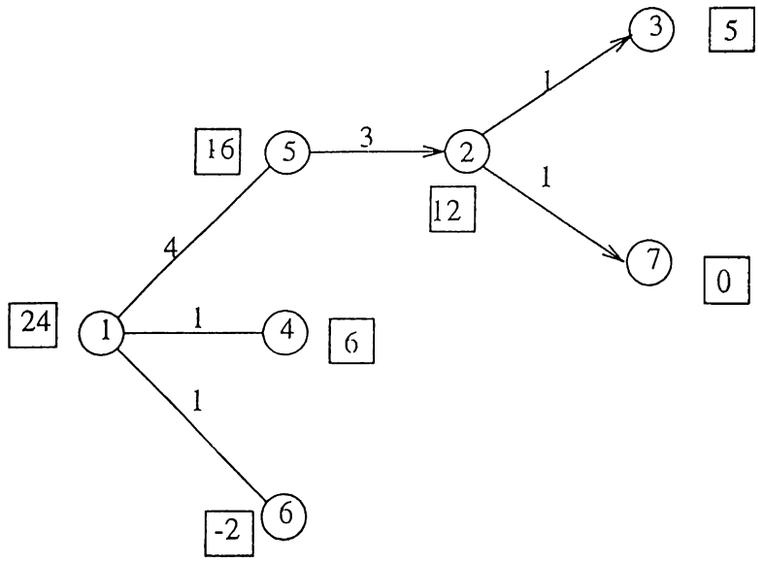
COST MATRIX :

∞	15	21	18	8	26
12	∞	7	10	23	20
15	17	∞	9	13	21
15	8	19	∞	11	25
22	4	12	23	∞	25
30	27	29	5	25	∞

We start the procedure with the shortest path tree T_0.

$$\sum_{(i,j)\in A} x_{i,j} < \begin{pmatrix} n+1 \\ 2 \end{pmatrix} = 21$$

T_0

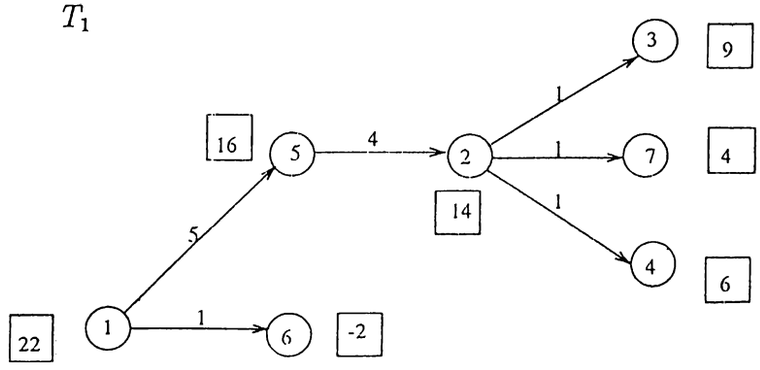


ITERATION 1: $\sum_{(ij)\in A} x_{ij} = 11 < 21$

$$\min \frac{\bar{c}_{ij}}{C^+_{ij} - C^-_{ij}} = \frac{\bar{c}_{24}}{C^+_{24} - C^-_{24}} = 2 \qquad \Delta\lambda = 2 \ , \ \lambda = 2$$

ENTER (2,4) , (1,4) LEAVES

T_1

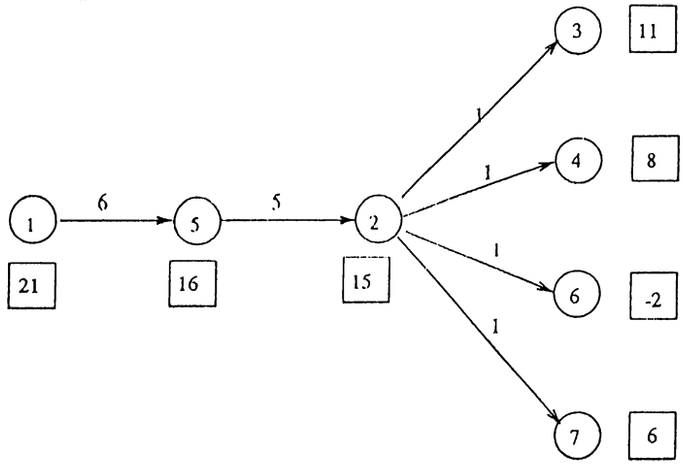


ITERATION 2: $\sum_{(ij)\in A} x_{ij} = 13 < 21$

$$\min \frac{\bar{c}_{ij}}{C^+_{ij} - C^-_{ij}} = \frac{\bar{c}_{26}}{C^+_{26} - C^-_{26}} = 1 \qquad \Delta\lambda = 1 \ , \ \lambda = 3$$

ENTER (2,6) , (1,6) LEAVES

T_2

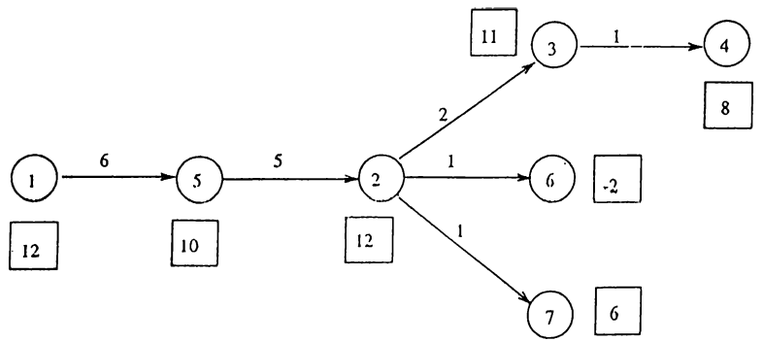


ITERATION 3: $\sum_{(ij)\in A} x_{ij} = 15 < 21$

$$\min \frac{\bar{c}_{ij}}{C_{ij}^{+} - C_{ij}^{-}} = \frac{\bar{c}_{34}}{C_{34}^{+} - C_{34}^{-}} = 3 \qquad \Delta\lambda = 3, \lambda = 6$$

ENTER (3,4) , (2,4) LEAVES

T_3

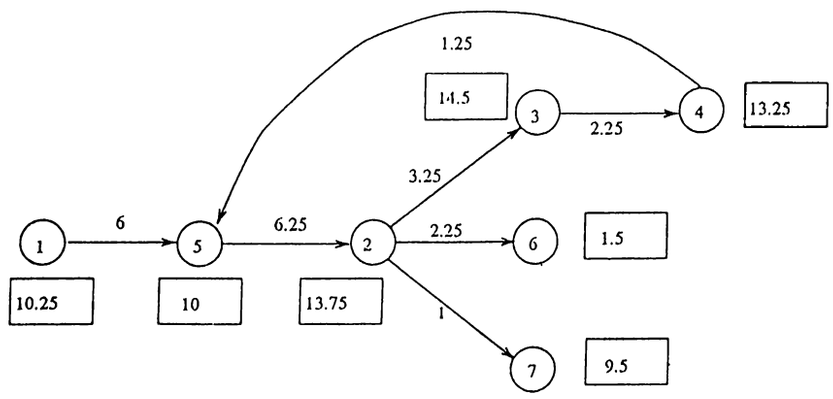


ITERATION 4: $\sum_{(ij)\in A} x_{ij} = 16 < 21$

$$\min \frac{\bar{c}_{ij}}{C_{ij}^{+} - C_{ij}^{-}} = \frac{\bar{c}_{45}}{C_{45}^{+} - C_{45}^{-}} = 1.75 \qquad \Delta\lambda = 1.75, \lambda = 7.75$$

ENTER (4,5) , THIS IS A CYCLE ! , AUGMENT (21-16)/4=1.25 ALONG THE CYCLE

T_4



3.3 Polyhedral Approaches

The set of feasible solutions of a linear programming problem $\{\min cx | Ax \geq b\}$ forms a polyhedron, i.e. the feasible region is the intersection of finitely many halfspaces. To apply the linear programming techniques, polyhedra have to be given in the form $\{x | Ax \leq b, x \in R^n\}$. An inequality $a_i^T \leq b_i$ is called *valid* with respect to a set $S \in R^n$, if $S \subseteq \{x \in R^n | a_i^T x \leq b_i\}$. In combinatorial optimization, polyhedra is given in the form $P_I = \text{conv}\{x | x \text{ is a feasible integral solution}\}$, i.e. as the convex hull of finitely many integer points. So, it is a major problem to find the valid inequalities defining such a polyhedron.

A subset F of a polyhedron P is called a *face* of P if there exists a valid inequality $a_i^T x \leq b_i$, such that F=$\{x \in P | a_i^T x = b_i\}$. It is said that the inequality $a_i^T x \leq b_i$ defines F. Facet defining inequalities are the strongest valid inequalities, so they are of particular importance, since one wants to find inequality systems with as few inequalities as possible. But it is difficult to identify and enumerate all facets and for most of the integer programming problems (for example for TSP), we do not have a characterization of all facets.

If we know all facets of P_I, then we have a minimal inequality system describing P_I and the integer programming problem reduces to a linear programming problem. But for most of the combinatorial optimization problems, even this minimal system contains a very large number of inequalities, which makes it practically impossible to list all of them. Due to these considerations, the idea of 'cutting plane' s arose. In cutting plane procedures, one starts with a polyhedron containing the feasible region, i.e. with a relaxation of the original problem. The aim is to find facets, which are close to the optimum point. Addition of these facets to the relaxed formulation narrows the feasible region of the relaxed problem by 'cutting' it and hopefully brings us to the neighborhood of the optimum point. This is a finite procedure, because there is only a finite number of facets. But one usually builds in a stopping criterion, for example stops adding the cuts, when the increase in the objective function gets smaller than a specified percentage. If the current optimum value is not feasible for the original problem, it can still be used as a lower bound for a branch and bound procedure.

In this section we describe some valid inequalities of the DMP and look at some procedures, which use those valid inequalities.

3.3.1 Valid Inequalities

In section 3.2 we obtained a relaxation of the DMP. Starting from that relaxed solution we tried to find some valid inequalities to narrow the feasible region and to improve the corresponding lower bound.

As we mentioned earlier, the relaxed shortest path formulation gives a solution, which is a tree with one cycle only. If we look at this relaxed solution and compare it with a feasible solution of the original problem, we see that the relaxed solution violates the feasible one in two ways. First , in the feasible solution, the incoming flow to a node set is limited, and the outgoing flow from each node set W (except node n+1) is greater than or equal to $|W|(|W|+1)/2$. Second, in the feasible solution, the total flow in a set W, is less than or equal to $(|W|\text{-}1)$(incoming flow to the set W)$+\binom{|W|}{2}$, where W$\in$V 1,n+1}..

So, the two types of valid inequalities are :

$$\sum_{i\in V, j\in W} x_{ij} \leq \sum_{k=n-|W|+1}^{n} k \qquad \text{W} \in \text{V}\backslash\{1,\text{n+1}\} \qquad \text{(1.a)}$$

$$\sum_{i\in V, j\in W} x_{ji} \geq \frac{|W|(|W|+1)}{2} \qquad \text{W} \in \text{V}\backslash\{1,\text{n+1}\} \qquad \text{(1.b)}$$

$$\sum_{i\notin W, j\in W} x_{ij} \geq \Big(\sum_{i\in W, j\in W} x_{ij} + \frac{(|W|-1)|W|}{2}\Big) \div (|W|-1) \qquad \text{W} \in \text{V}\backslash\{1,\text{n+1}\} \qquad \text{(2)}$$

We do not need to write inequalities (1.a) and (1.b) for j=1, because they are

automatically satisfied after splitting node 1. Similarly we do not need to write the inequalities (1.a) and (1.b) for j=n+1, because (1.a) is already satisfied and (1.b) is not valid for j=n+1.

3.3.2 Lagrangean Relaxation

Consider the following problem :

$$z = \min cx$$

$$\text{s.t. } Ax \geq b \qquad \text{(complicating constraints)}$$

$$Bx \geq d \qquad \text{(easy constraints)}$$

$$x \geq 0 \text{ and integer}$$

If we drop the complicating constraints $Ax \geq b$ we obtain a relaxation, which is easier to solve than the original problem.

Now let us consider the problem LR(λ), for $\lambda \geq 0$:

$$z(\lambda) = \min_{x} cx + \lambda(b\text{-}Ax)$$

$$\text{LR}(\lambda) \qquad \text{s.t. } Bx \geq d$$

$$x \geq 0 \text{ and integer}$$

$$\lambda \geq 0$$

The problem LR(λ) is called the Lagrangean Relaxation of the original problem with respect to $Ax \geq b$. In LR(λ), the complicating constraints are included in the objective function with a penalty coefficient λ. Since λ is positive, violation of the complicating constraints will increase the objective function. In other words, for $\lambda \geq 0$ and x feasible,

$$\max_{\lambda \geq 0}\{z(\lambda) | Bx \geq d, x \geq 0, x \text{ integer}\} \leq \min_{x}\{cx | Ax \geq b, Bx \geq d, x \geq 0, x \text{ integer}\} \qquad (4.1)$$

The Lagrangean or Dual problem is to maximize $z(\lambda)$, i.e.

$$\text{(LD)} \qquad \max z(\lambda) \quad \lambda \geq 0 \qquad (4.2)$$

The solution $x(\lambda)$ of the relaxed problem $LR(\lambda)$ is optimal for the original problem, if the following three conditions (global optimality conditions) are satisfied:

(1) $z(\lambda) = cx(\lambda) + \lambda(b\text{-}Ax(\lambda))$

(2) $\lambda(b\text{-}Ax) = 0$

(3) $Ax \geq b$

In the shortest path formulation of the DMP, constraints (1.1) and (1.4) can be thought of as 'easy constraints', because they define the 'Shortest Path Tree Problem', which can be solved in polynomial time. We do not need (1.5) in the set of 'easy constraints', because any feasible solution for (1.1) and (1.4) is integral. Then the valid inequalities described in the previous section are the 'complicating constraints'.

To apply the idea of Lagrangean Relaxation, we must find an appropriate λ for each complicating constraint and solve the corresponding $LR(\lambda)$.

Balas and Cristofides(1979) described an algorithm for the asymmetric TSP, where they used a 'restricted' Lagrangean Relaxation approach based on the assignment problem (AP). The choice of the Lagrangean multipliers guarantees the continued optimality of the initial AP solution, thus eliminates the need for repeatedly solving AP in the process of computing multipliers. In our Lagrangean Relaxation approach, we used a similar rule as Balas and Cristofides for the choice

of λ.

Let $a^i x \geq b_i$ be one of the complicating constraints and let W $\in$ E be the set of arcs which violate that constraint. To obtain dual feasibility we have chosen λ_i, such that

$$\lambda_i = \min\{\bar{c}_{ij} : (i,j) \in W\}$$

But if we choose λ in this way, after a few steps some reduced costs turn out to be zero and we cannot choose a positive λ. This prevents further improvement on the value of the objective function of LR(λ), which is a lower bound for the original problem.

On the other hand, we started the Lagrangean Relaxation procedure with the shortest path tree solution, which is a very weak lower bound compared to the LP relaxation solution. Since we did not include constraint (1.2) of the natural shortest path formulation in section 3.1., the valid inequalities (1.a) and (2) are automatically satisfied by the relaxed solution and the set of complicating constraints consists of valid inequalities (1.b) only. Therefore in our limited experiment, the Lagrangean Relaxation solution turned out to be smaller than the LP relaxation solution in almost all cases.

3.3.3 Cutting Plane Procedure

The exact number of the facets of the TSP is still not known, but the number of 'known' different facets of the TSP polytopes are calculated by Grötschel and Padberg(1979) for n$\leq$120, where n denotes the number of cities. For n=50, the number of subtour elimination constraints is 0.5×10^{15} and the number of comb constraints is 10^{60}. If n=120 these numbers turn out to be 0.6×10^{36} and 2×10^{179} respectively. Since 1979, other classes of facet defining inequalities are identified. Due to the huge number of constraints, it is not possible to list all of them and solve the TSP as a linear programming problem. So, we must find a set of 'suitable' constraints to solve the TSP. The idea of finding the suitable set of

constraints gave rise to the 'cutting plane' approach.

Here we propose a cutting plane procedure which uses the valid inequalities described in section 3.1.1. The procedure starts with the LP relaxation of the shortest path formulation and at each step one valid inequality is added to improve the lower bound on the objective function of the DMP.

STEP 0 : Let P_0 be the the LP relaxation of the DMP. Solve P_0. Set i=0.

STEP 1 : Compute the incoming and outgoing flows for each node. Sort the incoming flows in decreasing order and the outgoing flows in increasing order.

STEP 2 : Check whether there is a violated valid inequality (1.a) or (1.b) in P_i. If there is, add this inequality to P_i and call the new problem P_{i+1}. Set i=i+1, solve the new problem and go to STEP 1. If there is no such inequality go to STEP 3.

STEP 3 : Check whether there is a violated valid inequality (2) in P_i. If there is, add this inequality to P_i and call the new problem P_{i+1}. Set i=i+1, solve the new problem and go to STEP 1. If there is no such inequality, go to STEP 4.

STEP 4 : Apply a branch and bound procedure using the current lower bound.

To find a set which violates inequalities (1.a) and (1.b) in STEP 2 we do not need to check all subsets of the node set. We compute the incoming flow and outgoing flow for each node and sort these flows in decreasing and increasing order respectively. Then we look at the k largest incoming flows and k smallest outgoing flows by starting with $k=1=|W|$ and increasing k until a violated inequality is found. To decrease computation time, we first check the inequalities (1.a) and (1.b) and then look for a violated inequality (2).

For the time being, we could not develop a rule or heuristic, which prevents us from checking all subsets of the node set to find a violated inequality (2) or to show that there is no such violated inequality. To increase computational

efficiency, we look at the subsets in a special order. We first check the subsets of size 2 and n-2, then 3 and n-3 etc. In this way, our computation time decreased about 50 percent.

3.4 Heuristics

3.4.1 Heuristics for General Graphs

By adding the valid inequalities to the relaxed formulation, we obtain a lower bound for the optimum solution. Before entering branch and bound, we need some good upper bounds to make the branch and bound procedure more efficient.

We looked at three different heuristics. Heuristic 1 uses the idea of the combinatorial solution procedure for LP relaxation which is discussed in section 3.2.1. The main idea of heuristic 2 comes from a TSP heuristic, but the entering arc choice is made by considering the different structure of the objective function of the DMP. Finally heuristic 3 uses the idea of shortest processing time(SPT) rule.

Heuristic 1 :

This heuristic is very similar to the combinatorial solution procedure for LP Relaxation discussed in section 3.2. , the only difference is in STEP 2. In the combinatorial solution procedure, we allow an arc to enter, even if it creates a directed cycle, whereas in this heuristic we do not allow an arc to enter, if it creates a directed cycle. This way of choosing the entering arc guarantees that the solution will be a Hamiltonian path, whereas in combinatorial solution procedure we may end up with a tree with one cycle only.

STEP 0: Find the shortest path tree on graph G', call it T_0. Let π be the vector of dual variables. Set i=0, $\pi_0 = \pi$, $\lambda = 0$.

STEP 1: If $\sum_{(i,j)\in A} x_{i,j} < \begin{pmatrix} n+1 \\ 2 \end{pmatrix}$ then go to STEP 2, else STOP.

STEP 2: Consider only the nonbasic arcs, such that addition of them to the tree T_i does not create a directed cycle, i.e. there must be a leaving arc. Let C_{ij}^+ be the number of arcs in the cycle (not directed) created by adding arc (i,j) to the tree T_i, which have the same direction as the arc (i,j). Let C_{ij}^- be the number of arcs in the cycle, which have reverse direction as the arc (i,j). Find $\frac{\bar{c}_{kl}}{C_{kl}^+ - C_{kl}^-} = min \frac{\bar{c}_{ij}}{C_{ij}^+ - C_{ij}^-}$, where $C_{kl}^+ - C_{kl}^- > 0$. Let (k,l) be the entering arc.

STEP3 : Set $\Delta\lambda = \frac{\bar{c}_{kl}}{C_{kl}^+ - C_{kl}^-}$

STEP 4 : Update π's in the following way : Find the levels of the nodes in tree T_i, such that the level of the root is 1, the level of a node adjacent to the root is 2, etc. Keep π_l constant.Set$\pi_j = \pi_j + \Delta\lambda(level(j) - level(l))$.

STEP 5 : Set $\bar{c}_{ij} = c_{ij} - \pi_i + \pi_j - \lambda$. Set $\lambda = \lambda + \Delta\lambda$. Set i=i+1.

STEP 6 : Add the arc (k,l) to the tree T_{i-1} and delete the arc (j,l) to obtain the new tree T_i.

STEP 7 : Update flows and go to STEP 1.

<u>Heuristic 2:</u>

STEP 0 : Start with the path P_0 : 1-(n+1). Set i=0.

STEP 1 : Find a node k* not present in path P_i and an arc (i*,j*) of the present path, such that

$c_{i^*k^*}(x_{i^*j^*}+1) + (c_{k^*j^*}-c_{i^*j^*})x_{i^*j^*} + y_{i^*} = \min c_{ik}(x_{ij}+1) + (c_{kj}-c_{ij})x_{ij} + y_i$

where y_i represents the length of the path from node i to node j.

STEP 2 : Entering arcs are (i*,k*) and (k*,j*). Leaving arc is (i*,j*). Update the path P_i to obtain the new path P_{i+1}. Compute y_j's for each node j in the new path P_{i+1}. If the new path does not contain all nodes in the node set, set i=i+1 and go to STEP 1.

In STEP 2, we try to choose the entering arcs in such a way that the additional cost is minimal. At each iteration, the length of the path is increased by 1 and finally we obtain a Hamiltonian path.

Heuristic 3:

STEP 0 : Set k=0. Set i_k=1. Set LENGTH=0. Set P_k={1}.

STEP 1 : If LENGTH is less than $|V|$-1, choose a node $j^* \neq n+1$, such that $c_{i_k j^*} = \min c_{ij}$, $j \notin P_k$. Set $i_{k+1} = j^*$, $P_{k+1} = P_k + \{i_{k+1}\}$. Set LENGTH=LENGTH+1 and k= k+1.

STEP 3 : If LENGTH=$|V|$-1, add the arc $(i_k,n+1)$ to the path P_k and STOP.

Another version of this heuristic is to begin with node n+1 and look at the incoming arcs. At each step we should choose the incoming arc with minimum cost. But in almost all problems this version gave worse solutions. This result was expected, because flows are decreasing when we go from node 1 to node n+1 on the Hamiltonian path, i.e. the choice of the first arc of the path affects the objective function much more than the choice of the last arc.

Heuristic 4:

STEP 0 : Apply the cutting plane procedure to the original problem and let LP_0 be the problem after the addition of the cutting planes. Set i_0=1, j=0 and PATH={1}. Let x be the solution of LP_0. Let n be the number of nodes in the original graph.

STEP 1 : Choose i_{j+1}, such that $x_{i_j i_{j+1}}$ is maximum for i_{j+1}=2,...,n, $i_{j+1} \notin$PATH and $i_{j+1} \neq n+1$ if j<n.

STEP 2 : Set $x_{i_j i_{j+1}}$=n-j. Set $x_{k i_{j+1}}$=0 for $k \neq i_j$ and $x_{i_j k}$=0 for $k \neq i_{j+1}$, i.e. fix these variables and call the new problem LP_{j+1}.

STEP 3 : Set PATH=PATH+$\{i_{j+1}\}$, j=j+1. If PATH$\neq${1,...,n+1} then solve LP_j to obtain the new solution x and go to STEP 1, else STOP.

This is again a greedy heuristic like heuristic 3, but this time, at each step we choose the arc, which has the maximum flow among other possible arcs in the solution obtained after the cutting plane procedure. We fix the flow on this arc to the maximum possible value, set the flow on the other possible arcs to zero and resolve the current LP after fixing the flow on these arcs.

In order to improve the heuristic solutions, we applied another heuristic. This heuristic takes the paths found by other heuristics as input and tries to improve the solution by changing the positions of the nodes on the path. The basic idea is similar to the k-opt procedure of Lin and Kernighan(1973).

STEP 0 : Let P_0 be the path, which is obtained as a heuristic solution. Let C_0 be the cost of P_0. Set MINCOST = C_0 and MINPATH = P_0.

STEP 1 : Choose a pair of nodes (i,j), $i \neq j$, i,j = 2,...,n. Replace the positions of these nodes on MINPATH to obtain a new path P. Calculate the cost C of P. If C<MINCOST, set MINCOST=C, MINPATH = P and FLAG=TRUE. Repeat this procedure for all pairs (i,j). Go to STEP 2.

STEP 2 : Choose two pairs (i,j) and (k,l), $j \neq k$, such that j follows i on MINPATH and l follows k. Replace the positions of these pairs on the path MINPATH to obtain the new path P. Calculate the cost C of P. If C<MINCOST, set MINCOST=C, MINPATH = P and FLAG=TRUE. Repeat this procedure for all pairs (i,j) . Go to STEP 3.

STEP 3 : Choose two triples (i,j,k) and (l,m,n), $k \neq l$, such that k follows j, j follows i and n follows m, m follows l on MINPATH. Replace the positions of these triples on MINPATH to obtain the new path P. Calculate the cost C of P. If C<MINCOST, set MINCOST=C, MINPATH=P and FLAG=TRUE. Repeat this procedure for all triples (i,j,k). Go to STEP 4.

STEP 4 : Choose a node j on the path MINPATH,such that $1 \neq j \neq n+1$. Let i be the node following node 1 in MINPATH and k be the node following node j. Let l be the node prior to node n+1 on the path MINPATH. Replace the positions of the subpaths from i to j and from k to l to obtain the new path P. Calculate the cost C of P. If C<MINCOST, set MINCOST=C,

MINPATH=P and FLAG=TRUE. Repeat this procedure for all j. Go to STEP 5.

STEP 5 : If FLAG=FALSE, STOP. If FLAG=TRUE, set FLAG=FALSE and go to STEP 1.

This heuristic improved the other heuristic solutions in almost all examples.

3.4.2 Heuristics for Tree Graphs

Here we propose two heuristics for the solution of the DMP on tree graphs. The second heuristic is more general than the first one and gives better results in most of the cases.

Heuristic 1 : No Backtracking Case

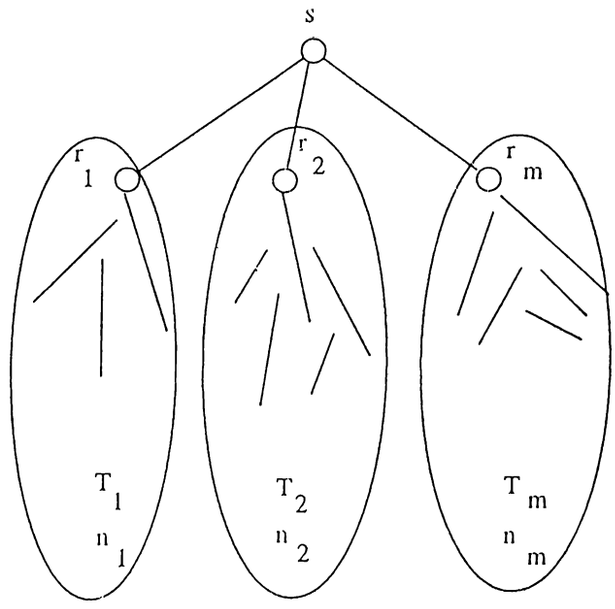
This heuristic is developed by generalizing the results related to star graphs. In this case, backtracking on the tree is not allowed.

STEP 1 : By starting from the vertices at the lowest level of the tree, calculate the weight w_k for each vertex except the root, where w_k is defined as the sum of the arc costs in the subtree with root v_k plus the travel time from v_k to its parent.

STEP 2 : Calculate the average weight for each v_k by dividing the weight by the number of vertices in the subtree with root v_k.

STEP 3 : If the current position of the delivery man is v_k, choose the vertex with smallest average weight for the next visit among the vertices, whose parent is v_k and which are not visited yet. If no such vertex exists, traverse the unique edge in the direction of the root and repeat STEP 3.

STEP 4 : Continue this procedure until all vertices are visited.



Suppose there are m subtrees connected to the root r, where subtree i has root r_i. Choosing vertex r_k for the next visit after r will imply to traverse all edges in subtree k before visiting any other vertex in the remaining subtrees, since no backtracking is allowed.

That means, the arrival times of all vertices in the remaining subtrees will increase by two times w_k.If the number of vertices in subtree i is n_i, in order to minimize the sum of the arrival times, subtree k should be visited before subtree i, if $w_k n_i < w_i n_k$. Since there are m main subtrees, subtree k should be visited before the others, if

$$w_k n_1 \leq w_1 n_k \quad \Leftrightarrow \quad w_k/n_k \leq w_1/n_1$$
$$w_k n_2 \leq w_2 n_k \quad \Leftrightarrow \quad w_k/n_k \leq w_2/n_2$$

$$w_k n_m \leq w_m n_k \quad \Leftrightarrow \quad w_k/n_k \leq w_m/n_m$$

where w_i/n_i is the average weight of r_i. This means that the vertex with the smallest average weight should be visited next and continuing this procedure

builds up heuristic 1.

Heuristic 2 : Backtracking Case

We showed that even if we choose the optimum depth first route, it may not minimize the sum of the arrival times. Sometimes it is possible to reach better results by backtracking. The second heuristic, which we developed, is to solve the TDTSP, when backtracking on the tree is allowed.

STEP 1 : Find the weights (w_k) and the average weights (a_k) of all vertices as described by the first heuristic.

STEP 2 : If the current position of the delivery man is at vertex j, let O_j be the set of vertices, which can be visited after vertex j.

STEP 3 : Let i^* be the index of the vertex with minimum average weight among the vertices in set O_j.

STEP 4 : Let $Q_j = O_j - i^*$

STEP 5 : Check whether there are vertices v_k in set Q_j, such that $c_{jk} < w_{i^*}/(L-1)$, where L is the number of vertices, which are not visited yet. If there are such vertices then let k^* be the index of the vertex with minimum c_{jk} .

STEP 6 : If a k^* is found in STEP 5, choose that vertex for the next visit and set $M = k^*$, else choose i^* and set $M = i^*$.

STEP 7 : Move from the current position to M, calculate the arrival time of M by adding the travel time from vertex j to M, to the arrival time of vertex j. Update z by adding the arrival time of M.

STEP 8 : If there are unvisited vertices, go to STEP 2.

There are two criterion taken into account by selecting the next vertex to visit. One of them is the average weights of the possible vertices for the next visit. This criteria is based on the fact that a depth first route may be taken on the subtree connected to the next visited vertex.

Second criteria is to consider the travel time from the current vertex to all possible vertices for the next visit. Even if the average weight of a vertex is high, the travel time to this vertex and back to the current vertex may be low, which allows economical backtracking. Suppose we have chosen i^*, which is the vertex with minimum average weight among the vertices in set O_j. By doing this, the arrival times of all vertices in set Q_j will increase by two times the weight of i^*, if we take a depth first route in the subtree with root i^*. If we visit any vertex with index k before visiting i^*, the arrival times of all unvisited vertices (except vertex k) will increase by the amount $2c_{jk}$, which is the cost of backtracking. In this case total increase in the value of the objective function will be $2(l-1)c_{jk}$. That is the reason why we check whether there is a vertex with $c_{jk} < (l-1)w_{i^*}$ which may allow economical backtracking.

3.5 Computational Results

To see the performance of the heuristics and the cutting plane procedure we did a computational study on complete graphs, where the costs are generated randomly.

For the cutting plane procedure, we have written a program in C-language, which uses the user subroutines of CPLEX interactively.

Table 1a and Table 1b show the results for complete graphs with 20 nodes, where cost ranges are 1-100 and 1-1000 respectively. We called the valid inequalities (1a) and (1b) as Type 1 cuts and the valid inequalities (2) as type 2 cuts. The results for graphs with 30 nodes can be seen in Table 2a and Table 2b.

The number of valid inequalities (2) is exponential and we could not develop a heuristic for choosing only a reasonable part of them. For the problems with 20 nodes addition of type 2 cuts could be done efficiently, but for the problems with 30 nodes much more time was needed. We also observed that after the addition of type 1 cuts, type 2 cuts did not increase the lower bound considerably. So, in

problems with 30 nodes we added type 1 cuts only.

It can be seen that there is a large difference between the tree solutions and the cutting plane solutions. Even the difference between LP-relaxation solutions and the cutting plane solutions is quite large. In some problems, the cutting plane solution is about four times the tree solution and two times the LP-relaxation solution (e.g. problems 2a20, 2b20, 7b20, 5a30).

In almost all examples, Heuristic 2 shows the worst performance. Compared to Heuristic 2 and Heuristic 3, Heuristic 1 and Heuristic 4 performed better. In some problems, the difference between the solution of Heuristic 4 and the solutions of other heuristics is very large(e.g. problems 1a20, 5a20).

We tried to find the optimal solutions of these problems, but neither LINDO nor CPLEX could solve them in reasonable time.

Chapter 4

CONCLUSION

In polyhedral approaches to the combinatorial optimization problems the main idea is to define the feasible region of the problem, which is usually given as the convex hull of finitely many integer points, with a linear inequality system.

In cutting plane procedures, the first step is to choose a suitable integer programming formulation of the problem and to obtain a lower bound by solving a relaxation of the problem. The next step is to find valid inequalities or 'cutting planes', which are preferably facets. Addition of these cuts to the relaxed problem will increase the lower bound by narrowing the relaxed feasible region.

Due to the difficult structure of the combinatorial optimization problems, it is usually very difficult to apply exact algorithms to large-scale problems. But most of the real life problems are large and it is important to find close-to-optimal and feasible solutions in a reasonable time. In such cases, heuristic or approximate methods are used.

Most of the combinatorial optimization problems are solved by using branch and bound methods. In branch and bound, subproblems are created by fixing the values of some variables and the optimal solution of the original problem is found by enumerating the points in the subproblem's feasible region. This enumeration

is done according to certain branching rules, which depend on the structure of the problem. In branch and cut methods, the valid inequalities are added to the subproblems and if a valid inequality is found at one branch, it can be added to the other branches, since it is 'valid' everywhere in the feasible region.

If we have a good lower bound and a good upper bound, the branch and bound or branch and cut methods can be applied more efficiently. The addition of the cutting planes narrow the search area and using the heuristic bounds we can 'prune' some of the branches. Cutting planes and heuristics together decrease the number of feasible solutions which should be checked and bring us closer to the region of the optimal solution.

In this thesis we proposed some polyhedral approaches to the Delivery Man Problem. Main idea was to find a lower bound for the problem using a cutting plane procedure and to find a good upper bound by trying different heuristics, so that a branch and bound or branch and cut procedure can be applied more effectively.

We gave two types of valid inequalities and proposed a cutting plane procedure and a Lagrangean Relaxation procedure based on these inequalities. We also proposed four heuristics for general graphs and two heuristics for tree graphs.

In our limited experiment, we have seen that the addition of type 1 cuts can be done in a reasonable time and it increases the lower bound considerably. Since we could not develop a rule for the addition of type 2 cuts, the addition of them takes a very long time compared to type 1 cuts. On the other hand, the addition of type 2 cuts increases the lower bound, which is obtained by adding all possible type 1 cuts, only by a small amount. Due to these observations, it seems more meaningful to enter a branch and bound procedure without adding type 2 cuts, unless a rule is developed for adding them in a reasonable time.

As a further research, a branching strategy can be found and new heuristics can be developed to improve the upper bound. The idea in Heuristic 4 can be enlarged and may be the starting point of a branching strategy. Also, new

valid inequalities can be found to increase the lower bound. In this thesis we worked with the natural formulation of the problem. Extended formulation can be studied in more detail and cutting planes for TSP can be translated to this problem using the extended formulation.

Bibliography

Balas E., Christofides N. (1979) A Restricted Lagrangean Approach To the Traveling Salesman Problem. *Carnegie Mellon University Management Sciences Research Report*, No. 439.

Dantzig G.B, Fulkerson D.R., Johnson S.M. (1954) Solutions of Large Scale Traveling Salesman Problem. *Operations Research*, No.2, pp.393-410.

Fox K.R. (1973) Production Scheduling on Parallel Lines with Dependencies. Ph.D. Dissertation, The Johns Hopkins University, Baltimore.

Fox K.R., Gavish B., Graves S.C (1980) A n-constraint Formulation of the (time-dependent) Traveling Salesman Problem. *Operations Research*, vol.28, No.4, pp.1018-1022.

Garey M.R., Johnson D.S. (1979) *Computers and Intractability, A Guide to the Theory of NP-Completeness*,Bell Telephone Laboratories.

Gavish B., Graves S.C (1978) The Traveling Salesman Problem and Related Problems, Working Paper GR-078-78, Operetions Research Center, Massachusets Institute of Technology.

Grötschel M. (1982) Approaches to Hard Combinatorial Problems. *Modern Applied Mathematics, Optimization and Operations Research*, North-Holland, pp.437-515.

Grötschel M., Padberg M.W. (1985) Polyhedral Theory, in the *The Traveling Salesman Problem*, Wiley, pp.251-306.

Hoffman A.J., Wolfe P. (1985) History, in the *The Traveling Salesman Problem*, Wiley, pp.1-16.

Johnson D.S., Papadimitriou C.H. (1985) Computational Complexity, in the *The Traveling Salesman Problem*, Wiley, pp.37-86.

Langevin A., Soumis F., Desrosiers J. (1990) Classification of Traveling Salesman Problem Formulations. *Operations Research Letters*, vol.9 ,No.2 , pp.127-132.

Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G., Shmoys D.B.,editors, *The Traveling Salesman Problem*, Wiley, 1985.

Lin S., Kernighan B.W.(1973) An Effective Heuristic Algorithm For the Traveling Salesman Problem. *Operations Research*, vol.21, No.2, pp.498-516.

Lucena A. (1990) Time-Dependent Traveling Salesman Problem-The Deliveryman Case. *NETWORKS*, vol.20, pp.753-763.

Miller C.E., Tucker A.W., Zemlin R.A. (1960) Integer Programming Formulation of Traveling Salesman Problems. *J. ACM* 7, pp.326-329.

Minieka E. (1989) The Delivery Man Problem on a Tree Network. *Annals of Operations Research*, Vol.18, pp.261-266.

Nemhauser G.L., Wolsey L.A. (1988) *Integer and Combinatorial Optimization*, Wiley.

Evans J.R., Minieka E. (1992) *Optimization Algorithms for Networks and Graphs*, Marcel Dekker Inc.

Padberg M.W., Grötschel M. (1985) Polyhedral Computations, in the *The Traveling Salesman Problem*,Wiley, pp.307-360.

Picard J.C., Queyranne M. (1978) The Time-Dependent Traveling Salesman Problem and Its Application to the Tardiness Problem in One-Machine Scheduling. *Operations Research*, vol.26, No.1, pp.86-110.

Schrijver A. (1986) *Theory of Linear and Integer Programming*, Wiley.

Simchi-Levi D., Berman O. (1991) Minimizing the Total Flow Time of n Jobs on a Network. *IIE Transactions*, vol.23, No.3, pp.236-244.