Imperial College of Science, Technology and Medecine

(University of London)

School of Management

DESIGN OF OPTIMAL DISTRIBUTION NETWORKS

by

Abdelghafor KHELIFA

A thesis submitted for the degree of

Doctor of Philosophy of the University of London

and for the

Diploma of Imperial College

March 1990

ABSTRACT

Network Design encompasses a broad range of configuration questions including the optimal structure of a distribution network and the best method of connecting a set of terminals to a central computer.

For our general network design model, each arc has a given length or "cost". The Capacitated Spanning Tree problem is one of the most important in the class of Network Design problems, where we have a specific traffic load associated with each vertex and a constraint associated with the total traffic carrying capacity of each arc. The objective is to find a minimum cost spanning tree connecting a given root vertex to the other vertices satisfying the constraint that the total traffic load in any subtree connected to the root does not exceed the arc capacity restriction. Two mathematical programming formulations of the above problem have been considered. The first one is a 0-1 mathematical programming formulation in two sets of variables. The solution of this formulation has gone through the development of a Benders Decomposition procedure where one iterates between two subproblems (a Master and a Slave), one for each set of variables. Difficulties that arose for getting the dual variables of the Slave subproblem have been overcome by theoretical developments from linear programming duality. Solutions for the Master subproblem required the development of a Lagrangean Relaxation bound for embedding into a tree search. Computational results on medium sized problems (up to 25 vertices) have proven satisfactory since only few iterations are needed for the algorithm to converge. The second formulation is also a 0-1 mathematical programming formulation in one set of variables. The difficulty of this formulation is that it involves a very large number of variables and reductions and exclusion tests are needed in order to reduce their number. The resulting problem can then be solved through the solution of a Set Partitioning Problem. Computational results with this formulation are given and medium-size problems (up to 26 vertices) can be solved optimally in reasonable time.

A mes parents sans qui tout cela n'aurait pas ete possible.

ACKNOWLEDGEMENTS

I would like to express my deepest thanks for my supervisor Professor Nicos Christofides for his guidance, help and encouragement throughout the course of this research.

I am also grateful to many of my fellow colleagues who helped me during my years of research with their kindness, help and advice. My particular thanks go to Dr A. Lucena and Dr. H. Cheddad.

The financial support of the Algerian Ministry of Higher Education is gratefully acknowledged.

I take the opportunity to thank my whole family for the constant help, understanding and encouragement during the process fo this research.

TABLE OF CONTENTS

ABSTRACT i

ACKNOWLEDGEMENTS ii

CONTENTS iii-v

CHAPTER 1 : Introduction 1

1-1 General 1

1-2 Outline of the thesis 2

1-3 Problem specification and NP completeness 3

1-3-1 Problem definition 3

1-3-2 The theory of NP completeness 6

1-4 Algorithms for the CST : A literature survey 12

1-4-1 Heuristics 12

1-4-2 Exact algorithms 23

1-5 Conclusions 43

CHAPTER 2 : 0-1 Integer Programming Formulation using Benders Decomposition 45

2-1 Introduction 45

2-2 Benders' decomposition scheme 46

2-2-1 Example 54

2-3 Problem formulation 58

2-4 Outline of the algorithm based on Benders' decomposition procedure 63

2-5 A heuristic for the CST 69

2-6 Comments 73

CHAPTER 3 : Slave and Master problem Solutions 75

3-1 Introduction 75

3-2 Computing the dual variables 76

3-2-1 Edmonds' algorithm 80

3-2-2 Deriving the dual variables 90

3-3 Master problem solutions 101

3-3-1 Cutting plane algorithm for the Master problem 102

3-3-2 Tree search algorithm for the Master problem 105

3-3-3 Lower bounds to the Master problem 105

3-3-4 A Tree search based on a Lagrangean Relaxation bound 113

3-4 An upper bound for the Master Problem 120

3-5 Conclusions 121

CHAPTER 4 : Computational Results 122

4-1 Introduction 122

4-2 Numerical example 123

4-3 Data generation and description 134

4-4 Computational Results 136

4-5 Comments 146

CHAPTER 5 : A formulation for the CST based on Set Partitioning 147

5-1 Introduction 147

5-2 A 0-1 integer programming formulation for the CST 148

5-3 Relaxations and lower bounds for the CST 150

5-3-1 Lower bounds for the CST 150

5-3-2 Getting lower bounds on the values of the d_{il} 154

5-3-3 Generation of all feasible subtrees rooted at v_0 159

5-4 Reduction tests 163

5-5 Tree search for the CCSSP 173

5-6 Conclusions 175

CHAPTER 6 : Conclusions 177

REFERENCES 179

CHAPTER 1

INTRODUCTION

1-1 GENERAL

Some of the algorithms developed in the fields of Operations Research and combinatorial mathematics are directly applicable to Network Design. A Network Design problem is usually formulated as an optimization problem, where a total network cost must be minimized over network design variables defining the topological structure of the network.

Network Design includes problems such as : the optimal structure of a distribution network and the best method of connecting a set of terminals to a central computer. The Capacitated Spanning Tree problem (CST for short) is one of the most important in the class of Network Design problems.

1-2 OUTLINE OF THE THESIS

This thesis is concerned with the study of the Network Design problem known as the CST, and consists of two parts.

In the first part we give a general description of the problem being solved and report the techniques that have been proposed to deal with this problem. In the second part we describe new solution techniques based on 0-1 integer programming formulations of the CST. These algorithms are implemented and tested on a number of medium size problems.

In this Chapter we review the methods that have been proposed to date to solve the problem either by exact methods or by approximate algorithms.

Chapter 2 is concerned with the outline of a 0-1 integer programming formulation for the CST and gives a complete description of the Benders decomposition technique that is proposed to solve that particular formulation. A heuristic is also proposed for the CST.

In Chapter 3 an algorithm is proposed to derive the dual variables of the Slave problem in the Benders decomposition. Also, solution techniques are given for the Master problem.

Chapter 4 compiles all the computational results obtained in solving the CST by the method proposed in Chapter 2.

In Chapter 5 an alternative 0-1 integer programming formulation for the CST (based on the Set Partitioning Problem) is proposed together with the appropriate algorithms to solve it. Within that Chapter computational results are reported.

Finally, Chapter 6 contains the conclusions and recommendations for further work.

1-3 PROBLEM SPECIFICATION AND NP-COMPLETENESS

1-3-1 Problem definition

Consider a nondirected graph G = (V,A) where V = $\{v_1,v_2,...,v_n\}$ is a set of vertices and A is a set of arcs. In a communication network, vertex v_1 (the root) represents a central computer (concentrator or multiplexor) and vertices $\{v_2,v_3,....,v_n\}$ represent (n-1) terminals. We are given costs c(e) for each arc e in the set A. Our objective is to connect all the terminals with the vertex v_1 using paths of arcs with the minimum possible total cost. If we impose no other constraints on the topological structure, the problem reduces to the so called minimum spanning tree problem (MST).

A spanning tree is a connected graph without a circuit and Figure 1-1 shows an example of a spanning tree.

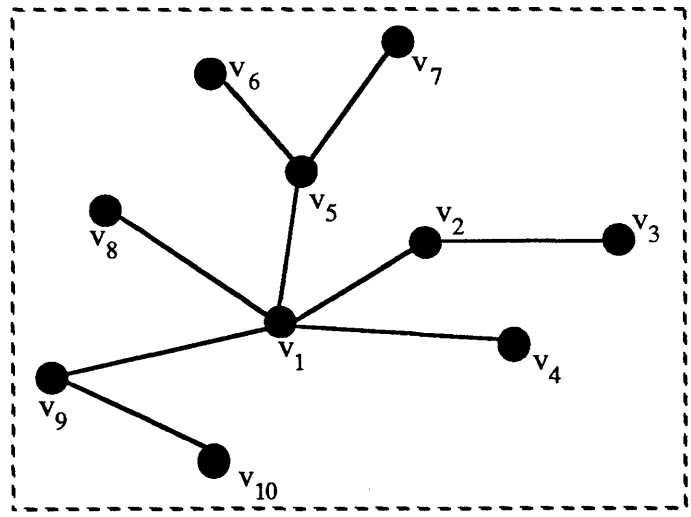


Figure 1-1 : A spanning tree.

The MST problem is one of several well-solved combinatorial optimization

problems. Various efficient (polynomial) algorithms have been developed for it : *KRUSKAL [100], PRIM [125].*

Kruskal's algorithm starts with a completely disconnected graph of n = |V| vertices. First we order the arcs in increasing order of cost c(e). We define a set T, which initially is empty, and will in general contain arcs in the MST solution. Add each successive arc to T provided this addition does not create a circuit with the other arcs in T. Repeat this step until T forms a tree containing (n-1) arcs. This final T is then an MST. The computational effort of the algorithm is dominated by the ordering step, for which a graph with m = |A| arcs would require O(mlogm) operations. If we assume a complete graph then O(mlogm) = $O(n^2 \log n)$.

Prim's algorithm produces an MST by expanding only one subtree until all vertices are connected. The tree T starts with an arbitrarily chosen vertex. T then grows by encompassing in T the nearest unconnected neighbouring vertex to one of the vertices already in T. After n-1 such steps, the tree T spans all the vertices and becomes an MST. It requires n-k comparisons to find the nearest neighbour at the k^{th} step, therefore, the total number of operations is $O(n^2)$.

If we have a requirement q(v) for each $v \in V - \{v_1\}$ and a capacity $\tilde{Q}(e)$ on each arc, then the general Capacitated Spanning Tree problem becomes :

Find a tree T that minimizes

$$\sum_{e \in T} c(e)$$

subject to :

$$\sum_{v \in U(e)} q(v) \leq \tilde{Q}(e)$$

where U(e) is the set of vertices whose path to v_1 contains arc e.

The above problem is referred to in the literature as the general Capacitated Spanning Tree problem. There are several variants of the problem, for example when we want to partition the set of vertices (excluding v_1) in, say, k = 1,...,M clusters in a such a way so that the total requirement load in any subset S_k does not exceed a given capacity Q_k as shown in Figure 1-2. In this problem the vertices of each cluster S_k are to be connected in a spanning tree to the root vertex via some "port" k. The "traffic" carrying capacity of this port is Q_k, which then becomes the restriction on the total requirement load of the cluster S_k. This variant of the problem is known simply as the Capacitated Spanning Tree problem and is referred to as the CST problem throughout this thesis.

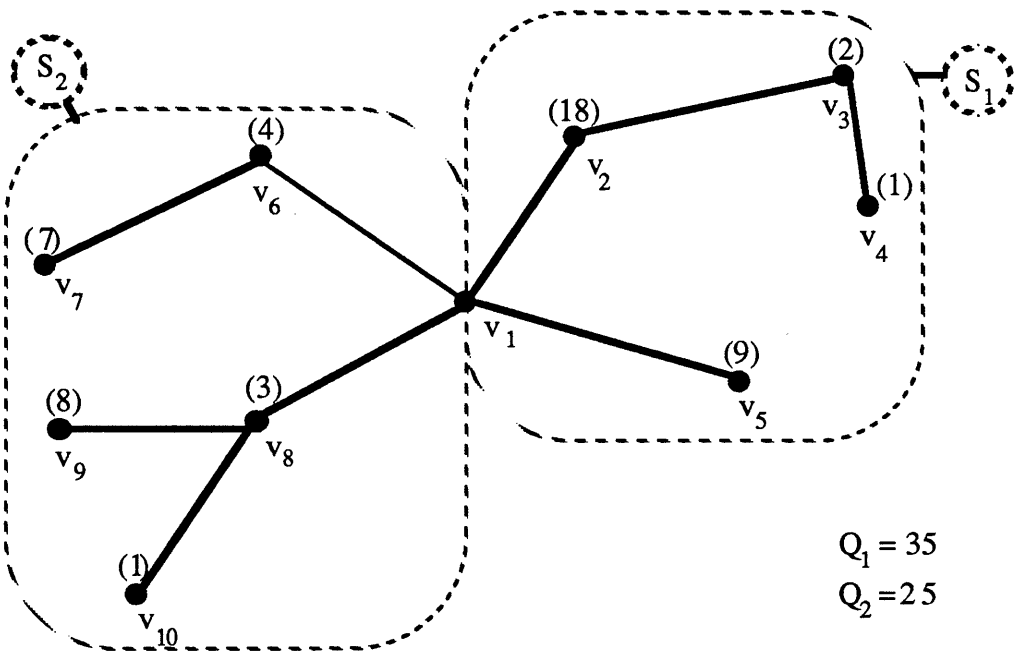


Figure 1-2 : Example of CST with M = 2

(The requirement load on each vertex is shown between parentheses)

Most of the research that has been done on the CST considered a simpler variant of the above problem where all q(v) = 1 , $\forall$ v $\in$ V and Q_k = Q , k = 1,...,M and M is not fixed in advance *[124]*. Some researchers, however, have considered the case where the q(v) are arbitrary but where Q_k = Q , $\forall$ k *[66]*.

Figure 1-3 shows a typical solution to a CST where $q(v) = 1$, $\forall\, v \in V$ and $Q_k = Q$, $\forall\, k$.

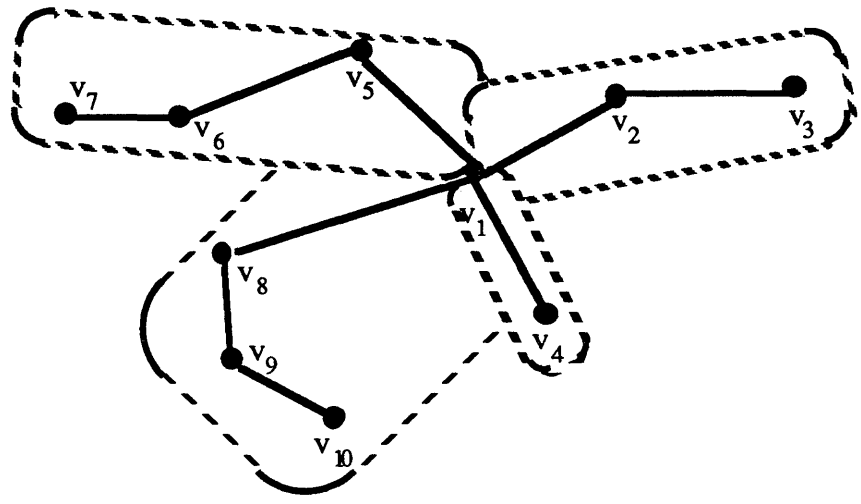


Figure 1-3 : Example of CST where $q(v) = 1, \forall\, v \in V$ and $Q = 3$

The CST problem defined above is much more difficult to solve than the (uncapacitated) MST problem. In particular, the greedy algorithms described earlier for the MST are no longer applicable to the CST and much more sophisticated and complex algorithms are required for the solution to the latter problem.
The addition of the capacity constraint destroys the nice structure of the MST problem and thus the two greedy algorithms described earlier cannot be applied.

1-3-2 The theory of NP completeness

All combinatorial problems fall into two main categories. The first category which consists of all undecidable problems has been known only for the last 50 years. In 1936 *TURING [139]* proved that some problems are so "hard" that no algorithm can ever be given to solve them, i.e. they are undecidable.
The second category which represents the vast majority of problems consists of all

the decidable problems.

Let us at this point introduce some basic concepts of problem complexity.

Concept 1 :

The way programs run on actual computers is that at any time, whatever the algorithm is doing, there is only one thing it could do next.

Such computers (and algorithms) are called deterministic computers (algorithms).

Concept 2 :

Because of its usefulness in understanding the theory of NP completeness, the concept of non-determinism was introduced. That is, to assume there exists a computer with the power that when an algorithm (that runs on this computer) is faced with a choice of several options, it has the capability to "guess" the right one. This is of course an unreasonable and unrealistic assumption.

Concept 3 :

Given a problem of size n (in our case n is the number of vertices in the network) and a real function t(n), we say that an algorithm runs in time O(t(n)) if the execution time of the algorithm f(n) is such that :

$|f(n)| \leq c\,|t(n)|$ where c is a constant and |f(n)| represents the absolute value of f(n).

For example if t(n) is polynomial in n we say that the algorithm runs in polynomial time. Prim's and Kruskal's algorithms for solving the MST are called "efficient" algorithms because their running time is polynomial as reported earlier.

The class of decidable problems can be divided into two subgroups. The first subgroup encompasses all problems that cannot be solved in polynomial time even with a non deterministic computer. This subgroup was first discovered in the 60's *[62]*. As far as the level of difficulty in solving the problems is concerned, this

category of problems comes directly after the class of undecidable problems.
The second subgroup of decidable problems is made up of all problems that can be solved in polynomial time with a non deterministic machine. It's called the NP class (NP stands for non deterministic polynomial). Most of the practical known problems and for this matter most of the combinatorial problems are non deterministically polynomial. We will restrict our attention to this subgroup of problems i.e. the NP class.

The most obvious question that arises now is to ask how "hard" (or how "easy") it is to solve a problem in the class NP. Although there is no rigorous or exact definitions for either term, computer scientists are of the view that a problem is considered to be "easy" if there exists an algorithm that can solve it in polynomial time on a deterministic machine.
To illustrate this, consider Table 1-1 which compares the performance of different time-complexity functions. Assuming that each computing operation takes one microsecond, each entry of the table gives the computing time of an algorithm that runs in O(f(n)) when it is applied to a problem of size n.
The first row of the table gives the size of the problem and the first column represents the complexity function. The conclusion from the table shows that as soon we pass from polynomially bounded problems to exponentially bounded problems it becomes impossible to solve even small size problems.
Looking at Table 1-1 one might argue that if we have a faster computer we can considerably reduce the execution time.

n f(n)	10	20	30	40	50	75	100	150
logn	2.3 microseconds	2.9 microseconds	3.4 microseconds	3.6 microseconds	3.9 microseconds	4.3 microseconds	4.6 microseconds	5.0 microseconds
n	10 microseconds	20 microseconds	30 microseconds	40 microseconds	50 microseconds	75 microseconds	100 microseconds	150 microseconds
n×*logn*	23 microseconds	60 microseconds	102 microseconds	147 microseconds	195 microseconds	323 microseconds	460 microseconds	751 microseconds
n^2	100 microseconds	400 microseconds	900 microseconds	0.0016 seconds	0.0025 seconds	0.0056 seconds	0.01 seconds	0.02 seconds
n^3	0.001 seconds	0.008 seconds	0.027 seconds	0.064 seconds	0.125 seconds	0.42 seconds	1 seconds	3.375 seconds
n^4	0.01 seconds	0.16 seconds	0.81 seconds	2.56 seconds	6.25 seconds	31.6 seconds	100 seconds	15.6 minutes
n^5	0.1 seconds	3.2 seconds	24.3 seconds	1.7 minutes	5.2 minutes	39.5 minutes	2.78 hours	39 hours
2^n	0.001 seconds	1.05 seconds	17.9 minutes	12.7 days	35.7 years	11.9×10^6 centuries	4.02×10^{14} centuries	4.5×10^{29} centuries

Table1.1: Comparison of running time of algorithms with different complexity functions.

Table 1-2 compares the performance of an exponential algorithm $O(2^n)$ when it is run on computers with different speeds. We can see that even if we devise a computer that is a million times faster than the current ones this will only produce a slight improvement in the size of the problem that can be solved in a given time by an exponential algorithm $O(2^n)$.

Computer	Current computer	Computer 10 times faster	Computer 100 times faster	Computer 1000 times faster	Computer 10000 times faster	Computer 100000 times faster	Computer 1 million times faster
Largest size problem	35	38	42	45	48	52	55

Table 1-2 : Performance of an exponential algorithm $O(2^n)$
when run on computers with different speeds.
(largest problem size that can be solved in 1 hour).

All the problems that can be solved in polynomial time by a deterministic algorithm are said to belong to the class P (for polynomial). They are "easy" and they also belong to the class NP (clearly if a problem can be solved in polynomial time by a deterministic machine then it can automatically be solved in polynomial time by a non deterministic machine).

A frequent question asked is whether P is equal to NP or not. This question has enabled researchers to produce a lot of work without answering the question. To show that P = NP one needs to prove that each problem of NP is in P, i.e. for each

problem of NP find a polynomial deterministic algorithm that can solve it. On the other hand to show that P ≠ NP one needs to exhibit a single NP problem which does not belong to P, i.e. for which no deterministic algorithm exists that can solve it in polynomial time.

It is a very frustrating situation to be in, especially when we know that a huge number of real life problems can be solved in polynomial time with a non deterministic algorithm, but for which no one has been able to produce a polynomial deterministic algorithm to solve one single such problem.

There is also a subgroup in the NP class that has the property that every member belonging to the NP class can be reduced by a polynomial algorithm to a member of that subgroup class. This class of problems is referred to as the NP-complete class.

The importance of this class is that if any one of its problems can be solved in polynomial time on a conventional (deterministic) computer, then so can all the other problems in the NP-complete class.

Proving that a given problem is NP-complete typically amounts to showing that this problem is in NP (by describing a polynomial non deterministic algorithm solving it), and then showing how a known NP-complete problem reduces to the given problem via a polynomially time bounded transformation (*COOK [39]*).

The list of NP-complete problems is a very long one headed by the very well known "satisfiability problem" (*GAREY and JOHNSON [62]*) and including the traveling salesman problem and the set-covering problem.

Now going back to our problem, the CST, falls in the NP-complete class (*PAPADIMITRIOU [124]*) and as a result there is little hope that "efficient" algorithms (i.e. polynomial algorithms) for its exact solution will ever be found.

1-4 ALGORITHMS FOR THE CST : A LITERATURE SURVEY

In the past there have been several attempts at the development of exact and approximate schemes for the solution of the CST. Several efficient heuristic algorithms were found to be successful in producing good solutions for problems of size ranging to hundreds of vertices, and some exact algorithms for the exact solution of the CST were applied with success to instances of size which to our best knowledge does not exceed 20 vertices.

We will be reviewing in this section the main heuristics and exact algorithms that have been proposed in the literature for solving the CST.

1-4-1 Heuristics

Due to the complexity of the CST problem and the fact that there is little hope that one can ever devise an efficient algorithm for its exact solution, researchers have looked at approximate schemes. Most of these approximation schemes exploit the following important property of the minimum spanning tree problem (MST).

THEOREM :

Let $T_1, T_2, ..., T_k$ be a set of components (subtrees) generated at some stage during the formation of an MST. We define the distance between T_i and T_j by :

$$D(T_i, T_j) = \min_{u \in T_i, v \in T_j} \{ d(u,v) \}$$

where d(u,v) is the cost of the arc connecting vertices u and v (If there is no such arc then $d(u,v) = \infty$).

For a chosen subtree T_r ($1 \leq r \leq k$), find the subtree T_s that is closest to T_r and let the corresponding arc be denoted by (u*,v*), that is :

$$D(T_r, T_s) = d(u^*, v^*) \quad , \quad u^* \in T_r \, , \, v^* \in T_s$$

Then the arc (u*,v*) must be in the MST.

The above property allows us to apply a so called greedy algorithm (*LAWLER [104]*) as in the KRUSKAL and PRIM algorithms. In the MST problem the greedy algorithm simply keeps adding to T the shortest arc at a given stage and need not backtrack or look ahead. This property is shared by a class of optimization problems known as matroidal problems (*LAWLER [104]*).

Most heuristics for the solution to the CST problem can be placed in one of the two classes :

1- Primal heuristics, where one starts with a feasible solution to the problem and improves it.

2- Dual heuristics, where one starts with an infeasible (but optimal subject to those subset of constraints that are satisfied) MST and then generates modifications to make the solution feasible.

The simplest heuristic to present is a slight modification of KRUSKAL's algorithm (*KRUSKAL [100]*). Recall that in this algorithm, arcs are selected in the order of their cost so long as they do not form a loop. This algorithm merges clusters of vertices. To satisfy the capacity constraint the following variant could be used for the simpler variant of the CST where $Q_k = Q$, $\forall$ k = 1,...,M. If any cluster reaches

a demand load of Q units, then connect that cluster to the central vertex by the shortest arc and remove the cluster from further consideration. Clusters are not connected if their combined load exceeds Q. This algorithm produces a solution that satisfies the constraint, and is as efficient in running time as the unconstrained MST algorithm, but does not necessarily result in an optimal constrained MST solution. Improvements to the above algorithm were proposed by several authors and in general yielded good solutions (that on average are between 5% and 10% away from the optimum *BOORSTYN and FRANK [22]*).

Esau-Williams' algorithm [48]

Esau and Williams introduced the notion of the weight of a vertex (w_i), which is defined as the distance from the closest member of the component containing vertex i to the root vertex (central facility). A tradeoff function t_{ij} is defined as follows :

$$t_{ij} = c_{ij} - \max \{w_i, w_j\}$$

The algorithm begins with ordering the t_{ij}'s in increasing order of their values, and the minimum t_{ij} is sought. Components corresponding to the minimum t_{ij} are merged if their union does not violate the capacity constraint and they do not form a cycle. By merging the two components a new component is formed. The affected w_i values are updated and the t_{ij} values are recomputed. The algorithm iterates in this fashion until a complete tree that encompasses the entire vertex set is formed.

The motivation for this procedure is to give preference to vertices which are far from the center and close together. Such vertices are difficult to treat in the sense that, if one waits too long, it may be necessary to connect them directly to the center using a long arc. Rigorously, one can think of this procedure as one starting with a star solution with all vertices connected directly to the center, and then transforming it by

replacing arcs connecting vertices directly to the center with shorter feasible arcs between them. The procedure is virtually identical to that called "savings" and first used for the vehicle routing problem by *CLARKE and WRIGHT [37]*, although in the original paper there is no reference to this fact.

Vogel's approximation method [140]

The Vogel approximation method (VAM for short) proceeds in a manner similar to the Esau-Williams algorithm with one difference. The t_{ij} are computed in the following manner :

$$t_{ij} = c_{ij} - \max(c_{ia_i} + c_{ib_i}, c_{ja_j} + c_{jb_j})$$

where a_i and b_i are the nearest and second nearest permissible (i.e. the two nearest vertices to the component containing vertex i which could be merged with that component without violating the capacity constraint) neighbours to the component containing vertex i and a_j and b_j are the nearest and second nearest permissible neighbours of the component containing vertex j. The algorithm seeks the smallest t_{ij} and merges the component containing vertex i with the component containing vertex j. As with the Esau-Williams algorithm, preference is given to potentially troublesome vertices.

Sharma's algorithm [133]

This heuristic exploits the radial nature of the geometry of vertices surrounding a center. This algorithm requires the use of the euclidean norm for the distances (or cost).

We express each vertex's location in polar coordinate, (r_i , θ_i) relative to the center. Starting from an arbitrary vertex, we sweep out an angle (with the center as a pivot) including vertices until the next one would violate the constraints. The set of vertices

is then connected to the center in an MST fashion. The sweeping procedure is then repeated starting with the first vertex left out, until the vertex set is exhausted. Thus, the network is divided into "pie" slices and vertices within a sector are connected in an MST. A refinement of this procedure which is slower in computational time (by a factor of n) but yields better solutions than the above procedure, is to successively use each vertex as the starting vertex and then (of the n solutions generated) picking up the best one. Improvements to solutions found with this algorithm can also be found via simple vertex exchange by moving individual vertices or groups from one sector to another neighbouring sector. This procedure is identical to one proposed earlier for the vehicle routing problem by *GILLET and MILLER [75].*

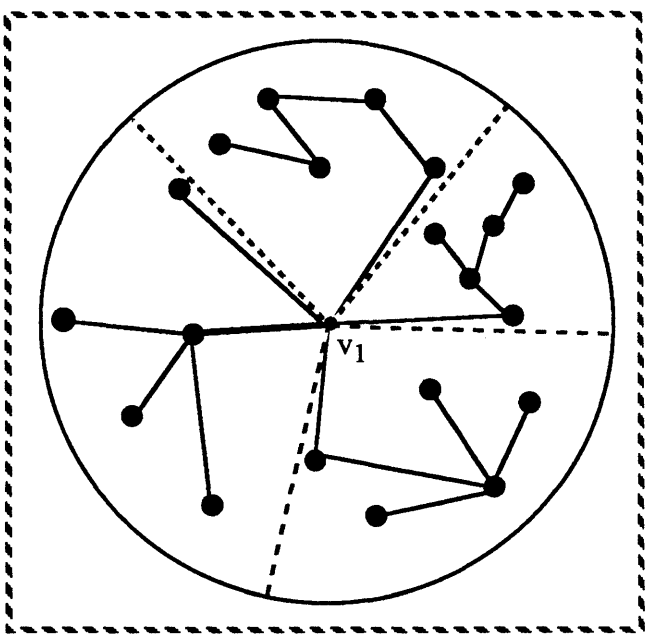


Figure 1-4 : Solution of Sharma's algorithm.

Martin's algorithm [118]

The vertex furthest from the center is chosen as a starting point and a tree is grown around this vertex, using Prim's algorithm until adding the next vertex would

violate the constraints. At this point, the tree is connected to the center by the shortest feasible arc. The procedure is repeated using the unconnected vertex furthest from the center and not part of any tree already found. This algorithm is also similar to one proposed for the Vehicle Routing Problem by *HAYES [79]*.

As we could notice all these heuristics mentioned above share the following properties :

1- Their running time is polynomial (of small order, $O(n^2)$, $p \leq 3$) in the number of vertices n and is independent of the capacity constraint.

2- The quality of the solution is not controllable and is not known even for the worst case (however computational results showed that some heuristics perform very well : within 10% away from the optimum).

Their inherent structure is sufficiently similar to allow them all to be implemented within a single unified framework. This has enabled one algorithm (*KERSHENBAUM et al [93]*) to generalize all the above heuristics and others and encompasses them in one general algorithm.

Each of the above algorithms starts with the vertices in different components and subsequently merges pairs of components. They differ only in the order in which they consider joining components. Since the grouping of a given set of vertices into one component restricts subsequent merging with other components because of the constraints, these algorithms, in general, yield different solutions. Thus, each algorithm starts by forming a different component and could in general yield a different solution from any of the others.

For example if we take the graph in Figure 1-5.

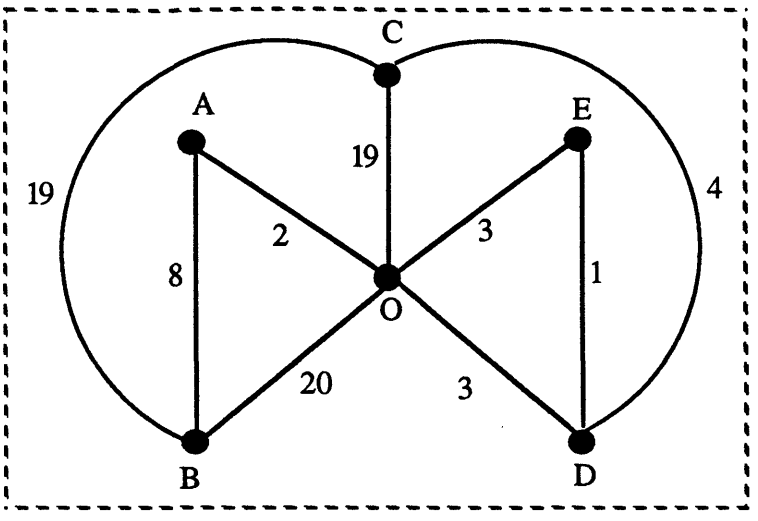


Figure 1-5 : Example graph

Prim's algorithm would consider arc (A,O) first.

Kruskal's algorithm would consider arc (D,E) first.

Esau-Williams' algorithm would consider arc (C,D) first.

Sharma's algorithm might consider arc (C,B) first.

The reason the algorithms choose different arcs is that they consider the vertices in a different order using the vertex weights to decide the order in which the components are to be considered for merging. Using this concept, Kershenbaum developed one algorithm which implements all the above heuristics and many others of the same type.

Kershenbaum's algorithm [93]

The following procedure will implement this class of algorithm given a set of rules which will be called w-rules, for initializing and updating the (implicit) vertex weights.

Step 0 Initialization.

Initialize w_i, i = 1,2,...,n using the appropriate w-rule.

Initialize the state vector V_i , i = 1,2,...,n containing the constraint information.

Set $C_i = i$, i = 1,2,...,n. C_i is the label of the component containing vertex i. Initially each vertex is in a separate component.

Set $S_i = 1$, i = 1,2,...,n. S_i is the size (number of vertices) of component i.

Set $t_{ij} = c_{ij} - \max\{w_i,w_j\}$ if arc (i,j) exists

$t_{ij} = \infty$ otherwise.

Goto Step 1.

Step 1 Get minimum entering arc.

Find $t_{i^*j^*} = \text{Min}\ (t_{ij})$.

If $t_{i^*j^*} = \infty$ terminate, otherwise Goto Step 2.

Step 2 Component merging.

Using the appropriate V_i evaluate the constraints on the component formed by merging the components containing vertices i* and j*. If any constraint is violated then set $t_{i^*j^*} = \infty$ and Goto Step 1, otherwise Goto Step 3.

Step 3 Arc admission.

Admit arc (i*,j*) to the spanning tree. Set $C_i = k$ for i in the components containing i* and j* i.e. merge the components containing i* and j* into a new component with label k. Set $S_k = S_{i^*} + S_{j^*}$ where S_{i^*} is the size of C_{i^*} and S_{j^*} is the size of S_{j^*}. Update the w_i using the desired w-rule and reevaluate the t_{ij}.

Calculate V_k, the state vector describing the newly formed component.

Goto Step 1.

Some remarks can be made :

- w-rules are used to assign weights to the vertices. They attach numerical values to the desirability of using an arc emanating from each of the vertices.
- t_{ij} is meaningful only if an arc (i,j) exists and the components containing vertices i and j can be merged without violating any constraints.

The following table shows how this algorithm can be reduced to the other algorithms mentioned earlier by choosing the appropriate w-weights.

ALGORITHM	INITIALISATION	Update when (i,j) is brought in
Prim	$w_1 = 0$ $w_i = \infty, \quad i = 2, ..., N$	$w_j = 0$
Kruskal	$w_i = 0, \quad i = 1, ..., N$	none
Essau-Williams	$w_i = c_{i\,1}, \quad i = 2, ..., N$	$w_j = w_i$ for $i \in C_i$
VAM	$w_i = c_{ib_i} + c_{ia_i}$	$w_i = c_{ib_i} + c_{ia_i}$

Table 1-3 : Comparison of different w-rules

KARNAUGH [90] has proposed a general extension to all the above primal CST heuristics which, at the expense of increased running time (in general by a factor of n^k), one could improve the quality of the solution obtained (empirically by an average of 3%).

In his paper Karnaugh defined two classes of algorithms :

- First order greedy algorithms (FOGA) which encompasses algorithms like Esau-Williams' and Martin's.
- Second order greedy algorithms (SOGA) which is a new class of heuristics.

In the SOGA algorithm one iterates one of the algorithms related above successively forcing selected arcs in or out the solution in order to obtain a sequence of solutions of improving quality. The increase in running time is a function of how many combinations of arcs are forced in and out of the solution. If one considers all potential combinations of arcs that can be forced in and out of the solution then we have a branch and bound procedure which then becomes an exact procedure.

Karnaugh's implementation relies on two procedures in order to decide which arcs are forced in or out of the solution.
These two procedures are described briefly below :

1- Inhibit procedure :

At each execution stage, FOGA algorithms bring in one arc connecting two previously unconnected sets of vertices. The Inhibit procedure successively prevents each of the mergers which occured in the previous iteration from taking place. Thus one iteration of the Inhibit procedure involves up to n iterations of the FOGA, each one of which prevents a single pair of vertex clusters from merging. At the end of each iteration of the Inhibit loop, the best of the generated solutions (and its associated cluster inhibitions) is kept and used in place of the original (uninhibited) FOGA solution, and the Inhibit procedure is repeated until no further progress can be made.

2- Join procedure :

For each vertex v_i find its nearest neighbour b_i and second nearest neighbour d_i among those which are closer to the center than v_i. Successively, one arc at a time, force arcs (v_i , b_i) and (v_i , d_i) into the solution if they are not already present. One Join loop consists of a sequence of executions of the FOGA, each with a single forced arc. The Join loop is iterated, starting from the best solution obtained during the previous iteration, until no further progress can be made.

In the second class of dual heuristics, one starts with (in general) an infeasible minimal cost solution to the problem and then iteratively modifies this solution, increasing its cost until a feasible solution is obtained.

Elias-Ferguson method [47]

Step 0 Generate an MST.

One starts with an MST, T. If T is feasible STOP, otherwise Goto Step 1.

Step 1 Arc swapping.

Repeat the following until T is feasible.

Replace arc (i,j) by arc (i',j') where (i,j) and (i',j') satisfy the following :

1- (i,j) is part of a subtree which violates the constraints.

2- (i',j') connects the component broken off by the deletion of (i',j') into a feasible subtree and the newly formed subtree violates the constraints by less than the subtree containing (i,j) did.

3- The difference in the costs of (i,j) and (i',j') is minimum over all arcs satisfying 1 and 2 above.

A variation which is somewhat slower but more effective is to allow the amount of infeasibility to increase if the exchange has not been previously tried while not allowing the cost to increase. This modified procedure is comparable in terms of its running time and effectiveness, to many of the primal algorithms, since each iteration either decreases the net cost or the infeasibility. This number of steps is a function of the constraint tightness and the center location in addition to the number of vertices. This procedure is closely related to simulated annealing (*KIRPATRICK et al [96]*). Thus the running time is less predictable than that of the primal heuristics.

Other dual heuristics algorithms have been proposed (*LIN [108]*, *SHARMA [133]*, *FRANK et al [58]*) where arc exchanges are used for transforming infeasible solutions once they are found. In a sense, all procedures mentioned thus far can be viewed in the very general context of arc exchanges procedures. Such procedures are powerful and general in scope. However, they rely heavily on two factors : one must either have a good starting solution or a method of restricting the selection of exchanges to be considered. No such insight is offered directly by the above procedures. All rely implicitly on being able to generate improvements with a very small number of arcs or at least being able to recognize the potential for improvements on the basis of a small number of exchanges.
Most primal heuristics mentioned earlier share the property that they are locally optimal with respect to a single arc exchange (such solutions are often called 1-optimal).

1-4-2 Exacts algorithms

There currently exist several techniques which all yield optimal solutions to the CST problem. These techniques can be divided into two main categories : branch and bound algorithms and integer programming based algorithms.

A) Branch and bound algorithms

These have been the most used techniques for solving the CST exactly. Branch and bound is a general technique for solving problems by the application of problem partition.
At the start we can easily derive upper and lower bounds to the optimum CST. The

upper bound is obtained by using one of the heuristics and a feasible but not necessarily optimum solution is obtained. The lower bound can be found by solving the MST problem, this solution will, in general, not satisfy the constraints. Next the solution space is subdivided into two or more subsets. For example in one, a particular arc is to be included in the final tree solution while in the other, that arc is forbidden. Upper and lower bounds are again derived for each subproblem. The best (feasible) upper bound is always recorded. If any of the lower bounds is greater than the best current solution no further subdivisions of that portion of the solution space would yield a better result. Therefore they need not be investigated. That leg of a branching tree is said to have been "fathomed". The process continues until no further branching is possible and the branch and bound should yield the optimum solution at the end.

CHANDY and RUSSEL [25] were the first to use a branch and bound technique to solve exactly the CST. The following is a small example to show how the technique works.

We are given a graph of 5 vertices with vertex 1 being the central facility (Figure 1-6).

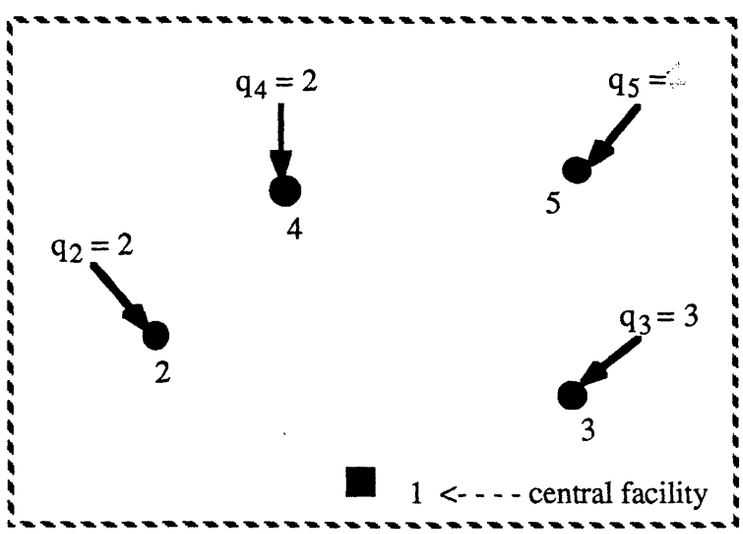


Figure 1-6 : Network example

The cost matrix of Figure 1-4 is given in Table 1-4.

$$C = \begin{bmatrix} - & 3 & 3 & 5 & 10 \\ 3 & - & 6 & 4 & 8 \\ 3 & 6 & - & 3 & 5 \\ 5 & 4 & 3 & - & 7 \\ 10 & 8 & 5 & 7 & - \end{bmatrix}$$

Table 1.4 : Cost matrix C.

The q_i are the capacities on the vertices, and the capacity constraint is that no more than 5 units flow on any arc is permitted.

The unconstrained MST is shown in Figure 1-7.

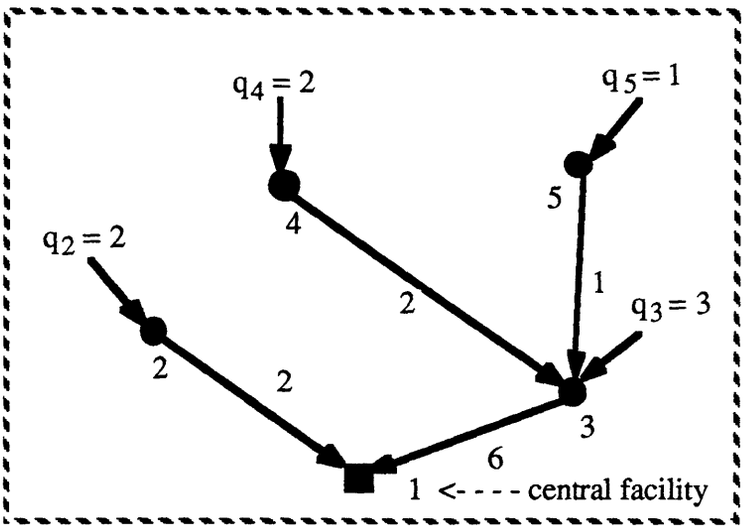


Figure 1-7 : Unconstrained MST of problem with cost matrix C .

In Figure 1-7 the numbers on the arcs are the flows running on the arcs and we can see that this tree is infeasible for the CST because arc (1,3) has got 6 unit flow on it

which exceeds the 5 units requirement.

The cost of this network is :

$$c_{21} + c_{31} + c_{43} + c_{53} = 14$$

which is a lower bound on the value of the optimum solution of the CST.

The algorithm of Chandy-Russel is based on the observation that those arcs in the unconstrained solution connected directly to the center remain connected to the center in the constrained solution.

In this example branch and bound works as follows : initialize the algorithm by partitioning the complete set of possible connections into subsets A and B. Let A contain the two arcs (1,2) and (1,3) and B have one or both of them lacking. Then clearly the constrained solution must be contained in A, with a lower bound cost of 14. The objective now is to partition A further, update the lower bound and check feasible solutions.

Let A be divided into two subsets AA and AB. To determine subset AA, let one of the tree arcs in A (one not already connected nor not allowed by constraint considerations) be connected. Subset AB then excludes this arc.

Thus subset AA consists of arcs (1,2), (1,3), (4,3) being established and no others arcs allowed. Subset AB consists of arcs (1,2), (3,1) being established and arc (3,4) not allowed.

Lower bounds are computed on these two subsets using the minimum spanning tree approach, which we will call L_{AA} and L_{AB}. We have then :

$$L_{AA} = 17 \text{ and } L_{AB} = 15.$$

The solution corresponding to $L_{AA} = 17$ is not feasible.

The solution corresponding to $L_{AB} = 15$ is feasible. Thus, this solution is also an upper bound to the problem, and AB is fathomed since $17 > 15$ and the tree search is shown in Figure 1-8.

The algorithm terminates with the optimum solution shown in Figure 1-9.

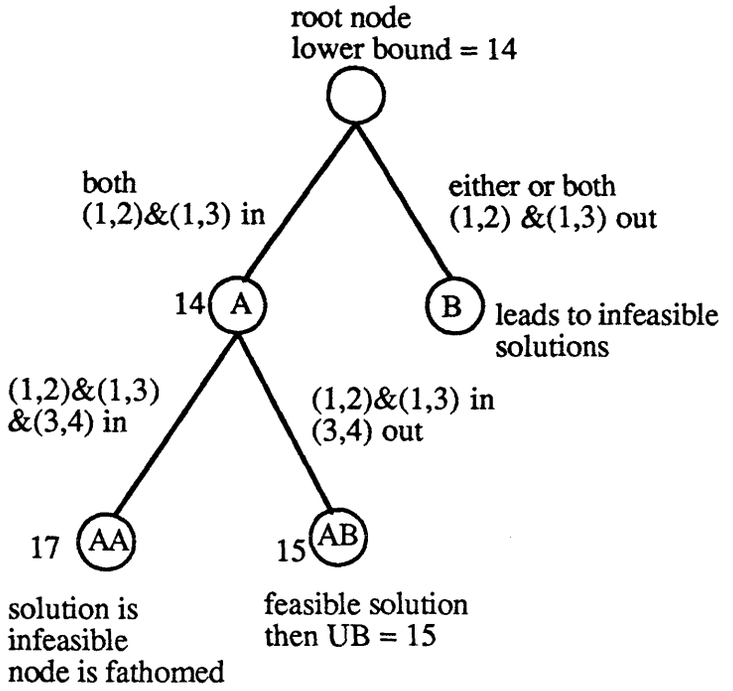


Figure 1-8 : Tree search for the example.

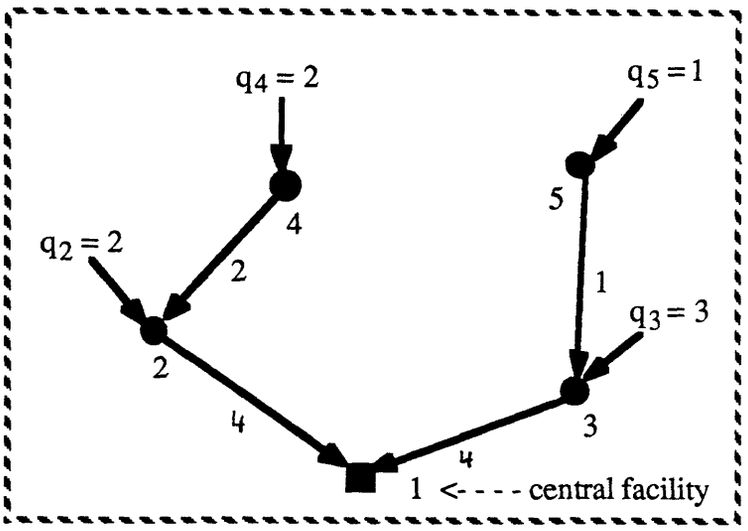


Figure 1-9 : Optimum solution of the CST with cost 15.

The major problem with branch and bound techniques is they are generally slow in execution time. However, other researchers have come up with different

versions of algorithms based on branch and bound techniques which improve on the results of the Chandy and Russel algorithm.

Chandy and Lo algorithm [26]

The original technique of Chandy and Russel was rather weak and has been subsequently refined by Russel and Lo. Here we give a brief description of this algorithm.

The algorithm is a modified version of Little 's procedure (*LITTLE et al [109]*). The cost of the minimum spanning tree is used as a lower bound on the cost of the CST. There is a partition of a set of solutions A into subsets B and B', where two vertices i and j are required to be in the same cluster in subset B and not in the same cluster in subset B'. A lower bound for B' is obtained by setting the costs of all arcs between the cluster that includes vertex i and the cluster that includes vertex j to infinity.

The refinement of this procedure over that of Russel and Chandy is to consider partitioning on the vertices rather on the arcs. By requiring and prohibiting vertex pairs from being in the same subtree one is implicitly requiring or prohibiting arcs to be in the same subtree, added with the observation that if the unconstrained spanning tree has arcs connected directly to the center then these arcs will be in the constrained spanning tree; this leads to a shorter running time of the algorithm as compared to the previous one.

Elias and Ferguson algorithm [47]

This is a modified version of the original Chandy and Russel algorithm. The partition is made on arcs with the following rules.

First only arcs connecting vertices to the center or arcs connecting vertices to vertices connected to the center by required arcs, are considered as candidates for being

required or prohibited. Furthermore, only arcs which are part of infeasible MST subtrees are considered.

By doing this, one simplifies the procedure because one knows immediately if one is forcing an infeasibility as one deals with clusters connected to the center. Also one only considers trees and does no checking for loops explicitly.

Finally, in the course of evaluating a subproblem, one does not have to regenerate much of the modified MST, as almost the entire modified MST of the immediate predecessor subproblem is the same. Indeed, in the case of requiring an arc, it is unaltered. In the case of prohibiting and arc, only one arc, that connects the component disconnected by the arc removal, need be recalculated.

The following observations were made by the authors :

a) If vertex v_i is part of an MST subtree rooted at v_f, and the MST arc (v_i,v_j) is prohibited, v_i will not be part of the subtree rooted at v_f.

b) Another observation is that if the constraint is that no more than m vertices be in any subtree, and there is a total of n vertices then there must be at least $K = \lceil n / m \rceil$ subtrees. Hence, any solution with fewer subtrees can be penalized by an amount equal to the minimum cost which must be added to create K subtrees.

The modified branch and bound algorithm is outlined below :

Each subset "s" is characterized by a set of "established arcs" (arcs contained in all feasible solutions in the subset) and a set of "disallowed arcs" (arcs excluded from all feasible solutions in the subset). Free arcs are those which are neither established nor disallowed. Established vertices are those connected to the center by established arcs and free vertices are the remaining vertices.

In each subset "s" the partial tree containing all the established vertices is called the established tree. Then the lower bound tree (LBT) of a subset s is defined as the unconstrained MST that connects with free arcs the free vertices to the established tree of the subset.

For partitioning a subset s we choose a free arc (i,j) from one of the arcs of the LBT that violates the constraints. The arc (i,j) becomes established in subset s_1 and

disallowed in subset s_2 as shown in Figure 1-8.

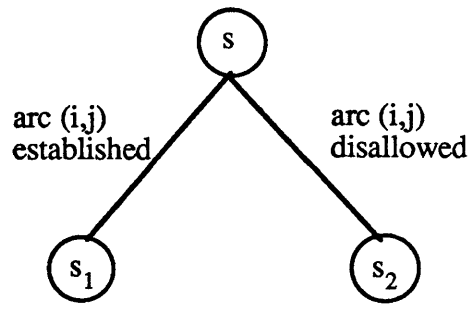


Figure 1-10 : Search Tree

Write a partition of k subsets as $\{s_1\text{-}b_1,...,s_k\text{-}b_k\}L$, where s_i is the i^{th} subset, b_i is its lower bound and L is the lowest lower bound.

Step 1 Compute an MST.

Find the MST. Let b_1 be the cost of the MST.

Step 2 Feasibility of the MST.

If the MST is feasible quit.

Step 3 Initial partitioning.

Form the subset s_1 by making all the central arcs of the MST established arcs. The partition is $\{s_1\text{-}b_1\}b_1$.

Step 4 Heuristic solution.

Use a heuristic algorithm to find a good feasible solution. Let F be the tree of the feasible solution. Let f_b be the cost of the feasible solution.

Step 5 Set the tree search to the first node i.e the root.

$i = 1$

Step 6 Partition of the current node.

Let the current partition be $\{s_1^{(i)} - b_1^{(i)},\ldots,s_k^{(i)} - b_k^{(i)}\}$ $L^{(i)}$

Pick the subset j with the lowest lower bound, i.e. $b_j^{(i)} = L^{(i)}$

If $L^{(i)} > f_b$ quit F is the optimal solution.

Step 7 Branching process.

Choose a free arc (q,r) from the arcs of a segment violating the constraints in the LBT of subset $s_j^{(i)}$ with $q \notin \varepsilon$, $r \in \varepsilon$.

Let $v^{(i)} = s_j^{(i)}$, $b_i = b_j^{(i)}$

Step 8 Partitioning.

Partition the subset $v^{(i)}$ into two subsets $s_j^{(i)}$ and $s_{k+1}^{(i)}$ by making the free arcs (q,r) established in $s_j^{(i)}$ and all the arcs (q,m) with $m \in \varepsilon$ and $f_m = f_r$ disallowed in subset $s_{k+1}^{(i)}$.

Step 9 Fathoming.

If $s_j^{(i)}$ does not contain any feasible solutions or if the established segments of $s_j^{(i)}$ do not have an MST topology, omit subsets $s_j^{(i)}$ from the partition, else compute $b_j^{(i)}$.

Step 10 Bounding procedure.

Compute the lower bound $b_{k+1}^{(i)}$ of the subset $s_{k+1}^{(i)}$.

If $b_{k+1}^{(i)} > f_b$ omit the subset $s_{k+1}^{(i)}$.

If the LBT of $s_{k+1}^{(i)}$ is feasible then let F = LBT, $f_b = b_{k+1}^{(i)}$, and

omit $s_{k+1}^{(i)}$ from the partition.

If the LBT is not feasible, then make all the central arcs of LBT established arcs.

Step 11 Pick a new node to branch on.

i = i+1 GOTO Step 6.

Kershenbaum and Boorstyn algorithm [95]

This algorithm is basically the same as the others algorithm described above but the difference is in the way the lower bounds are computed. In order to improve efficiency Kershenbaum and Boorstyn developed two algorithms, one based on generating subproblems by restricting vertices, and the other based on generating subproblems by restricting arcs. These techniques differ from previous ones in that the relaxation used here produces tighter bounds resulting in a smaller number of subproblems being generated.

We are only going to describe the relaxation scheme since the algorithm is the same in principle as the previous ones.

The CST examined by the authors in their paper is the problem of finding an MST subject to the constraint that the number of vertices (rather than the sum of loads of the vertices) in any subtree does not exceeds a prespecified maximum.

If we note that any optimal solution to the CST problem has the property that all subtrees are MSTs on the set of vertices contained in each subtree and the center, thus it suffices to find the optimal partition of the vertices into subtrees.

To find a partition of the vertices $V = \{ v_i / i=1,2,....,n\}$ into subtrees, we begin by making n copies of each vertex corresponding to the possibility of the vertex being in any of n possible subtrees. Thus v_{ij} corresponds to vertex v_i being in subtree j.

If we represent v_{ij} by a matrix :

$$W = \begin{bmatrix} v_{11} \ldots\ldots\ldots\ldots v_{1n} \\ v_{21} \ldots\ldots\ldots\ldots v_{2n} \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ v_{n1} \ldots\ldots\ldots\ldots v_{nn} \end{bmatrix}$$

Then the problem of partitioning the vertices into subtrees such that no subtree contains more than, say, m vertices can be viewed as one of selecting elements from W in such a way that no more than m elements are chosen in any column and exactly one element is chosen from each row. We want however to find an optimal partitioning i.e. a partition which gives rise to a CST of minimal total cost. If one is given a cost c_{ij} associated with each v_{ij} there exists an efficient technique (*LAWLER [104]*) for generating a partition of minimal total cost subject to the above constraint. Unfortunately no set of costs c_{ij} is currently known which corresponds directly to the cost of v_i being in subtree j. One can, however, define a set of costs c_{ij} which is a lower bound on the length of the optimal CST. Thus one can relax the CST problem into one of finding an optimal partition instead of a restricted MST as is done in other relaxations for branch and bound techniques.
This has proved to be a very successful relaxation since it results in a tighter bound being derived for the subproblems generated during the branch and bound process and consequently fewer branches are required in order to converge to the optimum.

B) Integer programming formulations

More recently a new direction in solving the CST has been taken by formulating the problem as an integer programming problem which is subsequently solved using the various available techniques.

GAVISH [66] developed the following formulation of the CST where the

capacity constraint is that no more than K vertices are required to be in any subtree.

Let :

$V = \{1,2,....,n\}$.

where 1 represents the central facility

Define a set of 0-1 variables x_{ij}, $i,j \in V$ such that :

$$x_{ij} = \begin{cases} 1 & \text{if arc } (i,j) \text{ is included in the minimal spanning tree} \\ 0 & \text{otherwise} \end{cases}$$

Define a set of variables y_{ij} , $i,j \in V$ that specifies the flow on the arc connecting vertices i and j .

The Capacitated Spanning Tree problem is formulated as follows :

$$\min \sum_{i=2}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \qquad (1)$$

subject to:

$$\sum_{j=1}^{n} x_{ij} = 1 \quad , \quad i = 1,...,n \qquad (2)$$

$$\sum_{j=1}^{n} y_{ij} - \sum_{j=2}^{n} y_{ji} = 1 \quad , \quad i = 2,...,n \qquad (3)$$

$$y_{ij} \le k \cdot x_{ij} \quad , \quad i = 2,...,n \quad j = 1,...,n \quad i \ne j \qquad (4)$$

$$x_{ij} \ge 0 \text{ and } y_{ij} \in \{0,1\} \quad , \quad \forall\, i,j \qquad (5)$$

The above formulation is based on a network flow formulation.

Summing up constraints (2) gives :

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

which is the required number of arcs of a spanning tree of an n vertex graph. Constraints (3) make the graph have no loops and constraints (4) puts an upper limit on the flow y_{i1} on each arc which is incident to vertex 1.

Given this formulation and the partitioning of the variables into two sets of variables, one [illegible] and the other integer, it was appealing to use Benders decomposition procedure to solve this formulation. Later on we will give a complete description of the Benders decomposition procedure when we will use it to solve another integer programming formulation of the CST.

The decomposition is straightforward. Given values for the assignment variables x^{*}_{ij} the Slave problem is a network flow problem given by :

Slave problem :

$$\min \sum_{i=2}^{n}\sum_{j=1}^{n} 0 \times y_{ij}$$

subject to :

$$\sum_{j\in A_i} y_{ij} - \sum_{j\in B_i} y_{ji} = 1 \qquad , \qquad i = 2,\ldots,n$$

$$0 \le y_{ij} \le K \qquad , \qquad \forall\ (i,j) \in A$$

$$y_{ij} = 0 \qquad , \quad \forall (i,j) \notin A$$

$$y_{ij} \in \{0,1\}$$

where :

$$A = \{(i,j) \ / \ x^{*}_{ij} = 1\}$$

$$A_i = \{j \ / \ (i,j) \in A\}$$

$$B_i = \{j \ / \ (j,i) \in A\}$$

The dual of the Slave problem is :

$$\max \sum_{i=2}^{n} U_i - \sum_{i=2}^{n} KV_{ij_i}$$

subject to :

$$U_i - U_j - V_{ij} \leq 0 \qquad , \quad j = 2,...,n \quad i \in B_j$$

$$U_i - V_{i1} \leq 0 \qquad , \quad i \in B_1$$

$$V_{ij_i} \geq 0 \qquad , \quad i = 2,...,n$$

From graph considerations to solve the above problem Gavish obtained the following Benders cut :

$$\sum_{i=2}^{n} U_i - \sum_{i=2}^{n} \sum_{j=1}^{n} KV_{ij} x_{ij} \leq 0$$

The Master problem is given by (1)-(2) supplemented by the set of cuts generated by the Slave problem.

The procedure iterates between two subproblems, the Master and the Slave problems, giving at each iteration two values which are respectively the lower bound and upper bound on the optimal solution to the original (CST) problem. Optimality is reached when the lower bound and upper bound are equal.

Unfortunately the computational results were disappointing since even for a small size problem (n=12) 900 Benders cuts were generated without reaching optimality. Taking into consideration that the Master problem is an integer program which is updated after each iteration and has to be optimally solved, the applicability of Benders decomposition to this formulation is severely limited.

A useful result from this formulation and the application of Benders decomposition was the identification of valid cuts for a linear programming characterization of the problem. It has been shown that the cuts derived are valid cutting planes in the linear programming relaxation of the problem.

Based on this work, *GAVISH [67]* developed a similar formulation for the capacitated minimal directed tree problem with a new Lagrangean Relaxation procedure for the problem in order to derive lower bounds on the optimal solution value.

The formulation is as follows :

Problem IP :

$$Z_{IP} = \min \sum_{i=2}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \qquad (6)$$

subject to :

$$\sum_{j=1}^{n} x_{ij} = 1 \quad , \quad i = 2,...,n \qquad (7)$$

$$\sum_{j=1}^{n} y_{ij} - \sum_{j=1}^{n} y_{ji} = 1 \quad , \quad i = 2,...,n \tag{8}$$

$$x_{ij} \le y_{i1} \le (K-1)x_{ij} \quad , \quad i = 2,...,n \quad j = 2,...,n \tag{9}$$

$$x_{i1} \le y_{i1} \le Kx_{i1} \quad , \quad i = 2,...,n \tag{10}$$

$$y_{ij} \ge 0 \quad , \quad x_{ij} \in \{0,1\} \quad , \quad i = 2,...,n \quad j = 1,...,n \tag{11}$$

adding the following constraints :

$$\sum_{i=2}^{n} x_{i1} \ge \left\lceil \frac{n-1}{K} \right\rceil \tag{12}$$

(where $\lceil x \rceil$ denotes the integer part of x)

which are redundant constraints to the above problem, one obtains the following relaxed problem when lagrange multipliers $\pi = \{ \pi_2, \pi_3, ..., \pi_n \}$ are attached to constraints (3).

problem L(π) :

$$L(\pi) = \min \sum_{i=2}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} + \sum_{i=2}^{n} \pi_i + \sum_{i=2}^{n} \pi_i \sum_{j=2}^{n} y_{ji} - \sum_{i=2}^{n} \sum_{j=1}^{n} \pi_i y_{ij}$$

subject to :

(7) , (9) , (10) , (11) and (12)

after rearrangement we have :

$$L(\pi) = \min \sum_{i=2}^{n} \pi_i + \sum_{i=2}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{i=2}^{n}\sum_{j=1}^{n} \beta_{ij}y_{ij} \qquad (13)$$

subject to :

(7) , (9) , (10) , (11) and (12)

where :

$$\beta_{ij} = \begin{cases} \pi_i - \pi_j & \text{for } i,j = 2,\ldots,n \\ \pi_i & \text{for } j = 1,\ldots,n,\ i = 2,\ldots,n \end{cases} \qquad (14)$$

One observation lead to show that :

$$y^*_{ij} \times \beta_{ij} = x_{ij} \times \gamma_{ij}$$

where γ_{ij} are defined as :

$$\gamma_{ij} = \begin{cases} \beta_{ij}(K-1) & \text{for } \beta_{ij} < 0 \\ \beta_{i1}K & \text{for } \beta_{i1} < 0 \\ \beta_{ij} & \text{otherwise} \end{cases} \qquad (15)$$

Thus substituting y_{ij} in (8) by x_{ij} leads to the following problem which involves x_{ij} variables only :

problem LR(π) :

$$LR(\pi) = \min \sum_{i=2}^{n} \pi_i + \sum_{i=2}^{n}\sum_{j=1}^{n} \tilde{c}_{ij}x_{ij} \qquad (16)$$

subject to :

$$\sum_{i=1}^{n} x_{ij} = 1 \quad , \quad j = 1,\ldots,n \qquad (17)$$

$$\sum_{i=2}^{n} x_{i1} \geq \left\lceil \frac{K-1}{n} \right\rceil \quad , \qquad j = 1,...,n \qquad (18)$$

$$x_{ij} \in \{0,1\} \quad , \qquad i = 1,...,n \; ; \quad j = 1,...,n \qquad (19)$$

where

$$\tilde{c}_{ij} = c_{ij} + \gamma_{ij} \qquad (20)$$

Using known results regarding the use of Lagrangean Relaxation in combinatorial optimization, one can shown (*FISHER [51]*) that $L(\pi)$ is a lower bound on Z_{IP}.

We are interested in finding the vector π^* which produces the tightest bound that is :

$$L(\pi^*) = \max_{\pi} L(\pi)$$

Analyzing problem $LR(\pi)$ we notice that if we drop the integrality constraint then the matrix is totally unimodular and thus the problem has got the integrality property that is :

$$LR(\pi^*) = L(\pi^*) = Z_{IP}.$$

where Z_{IP} is the optimal solution to the linear programming relaxation of problem IP. Therefore one way to obtain π^* is to solve the linear programming relaxation of problem IP. The other way is to use subgradient optimization to get π^*.

The computational results produced tight bounds with very little computational effort.

Another integer programming formulation for the CST was introduced by *GAVISH [68]*, which was based on the subtour elimination constraints type that have been successfully applied to the Traveling Salesman Problem (*BELLMORE et al [17]*). The CST under investigation is the problem of finding a minimum cost spanning tree, defined on an undirected graph G = (V,A), with the additional constraint that each subtree linked to the central vertex (root) has a total load that does not exceed Q.

As before G = (V,A) where V = {1,2,...,n} , 1 is considered as the central facility (root) and A is the set of arcs defined on the vertex set V. There is a load q_i associated with every vertex $i \in V - \{1\}$.

Let x_{ij} be binary variables defined as follows :

$$x_{ij} = \begin{cases} 1 & \text{if arc } (i,j) \text{ is used in the optimal solution} \\ 0 & \text{otherwise} \end{cases}$$

The integer programming formulation of the CST is then given by :

Problem IP :

$$Z_{IP} = \min\left[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} c_{ij}x_{ij}\right] \tag{21}$$

subject to :

$$\sum_{j=1}^{i-1} x_{ji} + \sum_{j=i+1}^{n} x_{ij} \geq 1 \quad , \quad i = 2,...,n-1 \tag{22}$$

$$\sum_{j=1}^{n-1} x_{jn} \geq 1 \tag{23}$$

$$\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} x_{ij} = n-1 \tag{24}$$

$$\sum_{i\in S}\sum_{\substack{j\in S \\ j>1}} x_{ij} \leq |S| - L_S \quad , \quad \forall\ S \subseteq \{2,3,...,n\},\ |S| \geq 2 \tag{25}$$

$$x_{ij} \in \{0,1\} \quad , \quad i = 1,...,n-1\ ,\ j = i+1,...,n \tag{26}$$

where $\tilde{L}_S = \max\{L_S,1\}$, and $\tilde{L}_S$ is the optimal solution to a one dimensional Bin Packing problem in which the bin length is Q and q_i, $i \in S$ are the lengths items that have to be packed in those bins. L_S is the minimal number of bins needed to pack the items in S.

Constraints (22)-(24) ensure that every feasible x corresponds to a tree. Constraints (25) guarantee that every subtree rooted at vertex 1 which is the central facility has a total load that does not exceed Q.

The problem was solved using an augmented Lagrangean based procedure like the one used successfully by *BALAS and CHRISTOFIDES [5]* for the asymetric Traveling Salesman Problem. This was done by relaxing constraints (25) and adding the following constraints :

$$\sum_{i\in S}\sum_{\substack{j\in S\\ j>1}} x_{ij} \leq |S|-1 \quad , \quad \forall\, S \subseteq \{2,3,...,n\},\ |S| \geq 2 \tag{27}$$

$$\sum_{j=2}^{n} x_{1j} \geq L_{V-\{1\}} \tag{28}$$

Some of the constraints of (27) are redundant in problem Z_{IP} but are relevant when constraints (25) are relaxed in a Lagrangean fashion using a vector $\pi = \{\pi_1,\pi_2,...\}$. The resulting problem $L(\pi)$ is as follows :

Problem $L(\pi)$:

$$L(\pi) = \min\left[\sum_{S\in\Psi}(|S|-L_S)\pi_S + \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} c_{ij}x_{ij} - \sum_{S\in\Psi}\sum_{i\in S}\sum_{\substack{j\in S\\ j>1}}\pi_S x_{ij}\right]$$

subject to :

(22)-24) and (26)-(28)

where Ψ denotes the set of unordered subsets of {2,3,...,n} with a cardinality that is greater than 1.

For a fixed vector of multipliers π, problem $L(\pi)$ is a Degree Constrained Minimum Spanning Tree problem (DCMST) in which the number of arcs adjacent to

vertex 1 (the root) is greater or equal to $L_{V-\{1\}}$. The cost function of the DCMST problem is given by :

$$\tilde{c}_{ij} = c_{ij} - \sum_{k \in \Psi_{ij}} \pi_k$$

where Ψ_{ij} is the index set of subsets of Ψ that contain both vertices i and j.

Clearly $L(\pi)$ is a lower bound on Z_{IP}. In order to have a tight bound one seeks a set of mulitpliers π^* that satisfies :

$$L(\pi^*) = \max_{\pi} \{L(\pi)\}$$

π^* can be best obtained by using Subgradient Optimization *[130]*.

The augmented Lagrangean procedure starts with a the set of lagrangean multipliers π set to 0 and then solve a DCMST porblem. The solution is checked and the constraints in (25) that are violated are identified and added to problem $L(\pi)$. Problem $L(\pi)$ is solved again and the procedure is repeated until either a solution is obtained that does not violate any of the constraints (25) or the whole procedure is stopped after a certain fixed number of iteration has been performed. The reported computational show that the bounds obatined were very tight and bettered those obatined in *[67]*. But one has to bear in mind that the above augmented Lagrangean procedure is very costly in terms of computing times since it relies on the solution of two known subproblems that are NP-complete, namely the Bin Packing problem and the Degree Constrained Minimum Spanning Tree problem. The resulting bound is meant to be imbedded in a tree search procedure.

1-5 CONCLUSIONS

In this Chapter we have reviewed some of the literature concerning the

solution for the Capacitated Spanning Tree problem (CST). There has been a significant amount of research carried out in this area in order to find acceptable solutions to the CST. These can be divided into two major groups. The first one is the group of algorithms that yield suboptimal solutions (heuristics) and they are all more or less based on variants of the algorithms that solve the Minimum Spanning Tree problem (MST). The second group of algorithms solves the CST exactly and can be divided into two subsets. The first kind of algorithm uses a technique often referred to as "branch and bound" where the solution space is enumerated implicitly and then by some tests reducing this solution space until it consists only of the optimal solution. The second type of algorithm consists in formulating the CST problem as an integer linear programming program, then using all the tools at hand in this area the problem is solved for the optimal solution. Some of the tools of interest are Lagrangean Relaxation and Benders' decomposition procedure, these will be used in the next Chapters.

CHAPTER 2

0-1 INTEGER PROGRAMMING FORMULATION USING BENDERS DECOMPOSITION

2-1 INTRODUCTION

In this Chapter we are going to present a new 0-1 integer programming formulation for the CST. This formulation decomposes into two subproblems, making it possible to use an elegant method devised by *BENDERS [18]* in the early 60's. Section 2-2 introduces the full Benders decomposition scheme, how to use it and why it is appealing to solve the CST in this way. In Section 2-3 we present a new 0-1 programming formulation in two sets of 0-1 variables. We will give a detailed outline of the algorithm devised, based upon Benders partitioning. Then we will focus our attention on how to solve the two subproblems which are the backbone of the Benders algorithm. In Section 2-5 we give an interesting new

heuristic for the CST and section 2-6 concludes the Chapter with comments on the material reviewed in it.

2-2 BENDERS DECOMPOSITION SCHEME

In the 60's Benders devised a very interesting procedure in which a partitioning approach is adopted for solving programming problems that involve a mixture of different types of variables or different types of functions. In general his approach can be used to solve the following mathematical programming problem :

$$\min cx + f(y)$$

$$\text{subject to :} \quad Ax + F(y) \geq b$$
$$x \geq 0 \, ; \, y \in Y \, .$$

Two common applications of the above formulation are in the solving of mixed integer linear programming and mixed linear and non linear problems. We are particularly interested in the case where f(y) is a linear combination of the y's and hence we have $f(y) = d \bullet y$, and in addition the set Y is the set of integer 0-1 y's. This kind of programs are referred to as mixed integer programs.

As applied to mixed integer problems, Benders' approach defines a totally integer problem that is equivalent to the original mixed problem and devises an iterative relaxation scheme for solving that integer problem.

Benders' partitioning approach

The general mixed integer problem may be stated as follows :

$$\min z = cx + dy$$

subject to : (P1)

$$Ax + By \geq b$$

$$x \geq 0 \; ; \; y \in \{0,1\}$$

The matrices A and B are $m \times n$ matrices .

$x , c \in R^n , y \in \{0,1\}^n \; b \in R^m$

For certain fixed values of y, problem (P1) is a linear program in x and it is natural to attempt to solve it by fixing y, solving a linear program in x, getting a "better" y and solving again a linear program, etc. One problem that arises is that only values of y for which x satisfies the resulting linear constraints may be considered. That is, we should have a set F from where to choose a feasible y. This set F can be written as follows :

$$F = \{ y / \exists x \geq 0 , \text{s.t.} : Ax \geq b - By ; y \in \{0,1\} \}$$

Any y belonging to F is a feasible y in the sense that the resulting linear program when y is fixed is not infeasible. Using Farka's lemma we can formulate explicitly the set of constraints of F. When y is fixed we have the following linear system :

$$Ax \geq b - By$$

$$x \geq 0$$

Adding a slack variable s we get :

$$Ax - s = b - By$$

$x \geq 0 , s \geq 0$

Farkas' lemma :

There exists a vector $x \geq 0$ satisfying $Bx = a \Leftrightarrow a'u \geq 0$ for all u satisfying $B'u \geq 0$

Applying this lemma to our linear system we get that y is feasible if and only if :

$$u(b-By) \leq 0 , u \geq 0$$

for u satisfying $uA \leq 0$

The cone $C = \{u / uA \leq 0 , u \geq 0\}$ is a convex polyhedron and it has a finite number of generators, i.e., there exist vectors $u_i^r, i=1,...,n_r$ such that any element $u \in C$ is a linear combination of the u_i^r i.e. :

$$u = \sum_{i=1}^{n_r} \lambda_i u_i^r \qquad , \lambda_i \geq 0$$

This enables us to rewrite the set F in the following manner

$$F = \left\{ y / u_i^r (b - By) \leq 0,\ i = 1,\ldots,n_r,\ y \in \{0,1\} \right\}$$

If F is empty then the original problem (P1) is infeasible. If the opposite holds we can rewrite (P1) as :

$$\min_{y \in F} \{dy + \min \{cx / Ax \geq b - By, x \geq 0\}\} \qquad (P1')$$

The inner optimization problem is a linear problem and is rewritten below together with its dual :

(Primal) :

$$\min cx$$

subject to : (P2)

$$Ax \geq b - By$$

$$x \geq 0$$

(Dual) :

$$\max u(b - By)$$

subject to : (DP2)

$$uA \leq c$$

$$u \geq 0$$

If $y \in F$, then the primal is feasible and by the duality theorem of linear programming we have :

$$\min \{ cx / Ax \geq b - By , x \geq 0 \}$$
$$= \max \{ u(b - By) / uA \leq c , u \geq 0 \}$$

Substituting this in (P1') we get :

$$\min_{y \in F} \{ dy + \max \{ u(b - By) / uA \leq c, u \geq 0 \} \} \quad (P1'')$$

Here we can note two interesting features of the feasible region $P = \{u / uA \leq c , u \geq 0\}$ of the inner maximization of (P1''). First the region P is independent of y. Second no matter what values of y are, the maximum of u(b - By) always occurs at a vertex of the convex polytope P (*DANTZIG [42]*), given the assumption that this polytope is bounded from above. Another remark can be made : the generators

of the cone C are the extreme rays of the polytope P since P and C define in fact the same region. Moreover, if P is empty then the primal (P2) is unbounded and so is the original problem (P1). If P is nonempty the inner maximization is achieved at a vertex of the convex polytope P or approaches $+\infty$ along an extreme ray of P.

There is a finite number of vertices of P , written as u_i^p, $i=1,\ldots,n_p$.

Then problem (P1") can be written :

$$\min_{y\in F}\left\{dy + \max_{i=1,\ldots,n_p} u_i^p(b-By)\right\}$$

This problem is equivalent to the following problem :

$$\min z$$

subject to :

$$z \geq dy + u_i^p(b-By) \qquad i = 1,\ldots,n_p$$

$$y \in \{0,1\}$$

Using the explicit form of the set F we can rewrite the above problem as :

$$\min z$$

subject to :

$$z \geq dy + u_i^p(b-By) \qquad i = 1,\ldots,n_p$$

$$u_i^r(b-By) \leq 0 \qquad i = 1,\ldots,n_r \qquad \text{(P3)}$$

$$z \geq 0\,,\ y \in \{0,1\}$$

The idea behind these transformations is to solve problem (P3) instead of problem (P1) (since they are equivalent) because of the fact that solving problem (P1) directly is very hard. The solution of (P3) using Benders' approach is elegant and it seems there are ways to get around the difficulties that arise when solving such integer programs as program (P3).

If we know the optimum vertex u* of :

$$\max_{i=1,\ldots,n_p} u_i^p(b - By) \quad , \quad \text{for any } y \in F$$

then we can solve (P3) as a "pure" integer program to obtain values z* and y* (The quotes around pure is to account for the fact that the program is not a pure integer program but contains one continuous variable z). Substituting the optimum y* into problem (P2) we solve the linear program :

$$\min z$$

subject to :

$$Ax \geq By^* \qquad (P2)$$

$$x \geq 0$$

and we obtain the optimum value x*.

Substituting x* and y* into the objective function of problem (P1) we get the optimal value $z = cx^* + dy^*$. This is of course the same value as the z* obtained in solving (P3).

The problem with this approach is that it is impossible to find the optimal u* because of the huge number of vertices of the polyhedron P and the impracticability of generating them all. If the polyhedron P is unbounded from above (and thus we need not worry about the problem of having to generate extreme rays of P), then problem (P3) will have in its linear system at least n_p constraints of the form :

$$z \geq dy + u_i^p(b - By)$$

where n_p is the number of vertices of the polyhedron P. Thus, if it is possible to generate all vertices of P, problems (P1) and (P3) are equivalent (in the sense that they both yield the same optimal value and solution we are seeking) and problem (P3) contains n_p constraints, one for each of the vertices of the polyhedron P. Therefore, starting with problem (P3) having only a few (generally one) of these constraints is a relaxation of problem (P3) and consequently of (P1). Problem (P3) being a minimization problem the solution to the relaxed problem of (P3) will yield a lower bound on the optimum value of (P1).

Starting from this simple idea Benders proposed the following iterative procedure to get the optimum u*. Take any feasible u^+ (not necessarily a vertex of P) that satisfies $uA \leq c$, $u \geq 0$ (i.e. $u^+ \in$ P), substitute it into (P3), i.e. start (P3) with only one constraint of the type :

$$z \geq dy + u^+(b - By)$$

Let the solution to (P3) be z^+ and y^+. Use y^+ in (DP2) and solve it. Let the solution be u^{++} (u^{++} is now a vertex of P). Put u^{++} into (P3), i.e. add one more constraint (or as it is often referred to in the literature : a Benders' cut) and solve for y again. If in the event of solving (DP2) we get an unbounded solution, then one adds to (P3) a constraint of the form :

$$u'(b - By) \leq 0$$

for that u' along which the maximum went to $+\infty$.

Now we formally state the partitioning algorithm :

STEP 0 Initialization.

Start with a $u^+ \geq 0$ that satisfies $uA \leq c$.

If none exists, then the original problem (P1) has no feasible solution.

STEP 1 Solve the Master problem.

Solve the "pure" integer program

min z

subject to : (P3)

$$z \geq dy + u^{+}(b - By)$$

$$z \geq 0 \, , \, y \in \{0,1\}$$

STEP 2 Solve the Slave problem.

Using y^{+} obtained in Step 1 , we solve the linear program :

max $u(b - By^{+})$

subject to :

$$uA \leq c$$

$$u \geq 0$$

If u goes to infinity then add to the problem (P3) the following constraint

$$u^{+}(b - By) \leq 0$$

else let the solution to (DP2) be u^{++}.

Determine whether :

$$z^{+} - dy^{+} \leq u^{++}(b - By^{+})$$

If the equality is satisfied Goto Step 3 . If not add the constraint :

$$z \geq dy + u^{++}(b - By)$$

to the existing set of constraints of (P3) and Goto Step 1.

STEP 3 Solve the original problem with y fixed.

Use y^{+} obtained in Step 1. Solve the linear program :

min cx

subject to :

$$Ax \geq b - By^{+}$$

$$x \geq 0$$

Let x^{+} be the solution .

Then x^{+} and y^{+} are the optimal solution of problem (P1).

The nice feature of this algorithm is that at any iteration we have a lower and upper bound on the optimal value of problem (P1). At Step 2, since not all the constraints of (P3) are included, the resulting optimal value of (P3) is a lower bound on the optimum value of (P1) and this value is monotonically increasing at each iteration. When y^{+} is known and at each iteration we solve the corresponding linear program (P2) and get a solution x^{+}, then $cx^{+} + dy^{+}$ is an upper bound to the optimal value of (P1) but is not necessarily monotonically decreasing. Therefore, some modifications to the above algorithm would make it possible to derive at each iteration a lower bound and an upper bound on the optimal value of (P1). The best lower bound is always achieved in the last iteration. The best upper bound is achieved by picking the best valued upper bound among those computed after a certain number of iterations.

2-2-1 Example

To illustrate how this algorithm works let us solve the following example :

$$\min 5x + 2y_1 + 2y_2$$

subject to :

$$\begin{aligned} x + 3y_1 + 2y_2 &\geq 5 \\ 4x - y_1 - y_2 &\geq 7 \\ 2x + y_1 - y_2 &\geq 4 \\ x \geq 0 \, , \, y_1, y_2 &\in \{0,1\} \end{aligned} \qquad \text{(I)}$$

Supposing that we have a fixed known y, then we rewrite the above problem (I) as :

$$\min\ 5x$$

subject to :

$$\begin{aligned} x &\geq 5 - 3y_1 - 2y_2 \\ 4x &\geq 7 + y_1 - y_2 \\ 2x &\geq 4 - y_1 + y_2 \\ x &\geq 0 \end{aligned} \qquad \text{(II)}$$

The dual of the above problem is :

$$\max(u_1,u_2,u_3)\begin{bmatrix} 5 - 3y_1 - 2y_2 \\ 7 + y_1 + y_2 \\ 4 - y_1 + y_2 \end{bmatrix}$$

subject to : (III)

$$u_1 + 4u_2 + 2u_3 \leq 5$$

$$u_1 , u_2 , u_3 \geq 0$$

STEP 0 Initialization.

Get a feasible solution u to the dual problem (III). One feasible solution to (III) is :

$$u_1 = 0 , u_2 = 5/4, u_3 = 0$$

i.e. this solution verifies (not necessarily optimal to (III))

$$u_1 + 4u_2 + 2u_3 \leq 5$$

1st iteration :

STEP 1 Solve the Master problem.

Rewriting (I) we have the following Master problem :

$$\min z$$

subject to :

$$z \geq 2y_1 + 2y_2 + \max_{\substack{\text{over all vertices} \\ u^p \text{ of } u_1+4u_2+2u_3 \leq 5}} (u_1^p + u_2^p + u_3^p) \begin{bmatrix} 5 - 3y_1 - 2y_2 \\ 7 - y_1 - y_2 \\ 4 - y_1 + y_2 \end{bmatrix} \qquad \text{(IV)}$$

$$z \geq 0 \, , \, y_1 \, , \, y_2 \in \{0,1\}$$

Substituting u = (0,5/4,0) in (IV) we have :

$$\min z$$

subject to :

$$z \geq 2y_1 + 2y_2 + \frac{5}{4}(7 + y_1 - y_2)$$

$$z \geq 0 \, ; \quad y_1 \, , \, y_2 \in \{0,1\}$$

The solution of this problem is :

$$z^+ = \frac{35}{4} \, , \; y_1^+ = y_2^+ = 0$$

STEP 2 Solve the Slave problem.

Substituting $y_1^+ = y_2^+ = 0$ into (III) we have :

$$\max \; 5u_1 + 7u_2 + 4u_3$$

subject to :

$$u_1 + 4u_2 + 2u_3 \leq 5$$

$$u_1 \, , \, u_2 \, , \, u_3 \geq 0$$

The solution is u = (5,0,0) with value 25.

Since we have :

$$\frac{35}{4} - [22]\begin{bmatrix}0\\0\end{bmatrix} \leq 25$$

and the equality is not satisfied we perform another iteration.

2nd iteration :

We add another constraint (Benders' cut) to (IV).

STEP 1 Solve the Master problem.

min z

subject to :

$$z \geq 2y_1 + 2y_2 + \frac{5}{4}(7 + y_1 - y_2)$$

$$z \geq 2y_1 + 2y_2 + 5(5 - 3y_1 - 2y_2)$$

$$z \geq 0 \, , \, y_1 \, , y_2 \in \{0,1\}$$

The solution to the above problem is :

$$y_1 = 1 \, , \, y_2 = 0 \text{ with } z = 12$$

STEP 2 Solve the Slave problem.

Substituting $y_1 = 1$ and $y_2 = 0$ into (3) we have :

$$\max 2u_1 + 8u_2 + 3u_3$$

subject to :

$$u_1 + 4u_2 + 2u_3 \geq 5$$

$$u_1 \, , u_2 \, , u_3 \geq 0$$

This yields a solution as :

$u_1 = 5 \, , u_2 = u_3 = 0$, with value of the objective function equal to 10

Since :

$$10 = z - [22]\begin{bmatrix}1\\0\end{bmatrix} = 12 - 2 \times 1 - 2 \times 0 = 10$$

The solution obtained is optimum and the algorithm terminates.

STEP 3 Solve original problem with the y of the last iteration.

We get the optimum x by substituting $y_1 = 1, y_2 = 0$ into (III) :

min 5x

subject to :

$$x \geq 2$$

$$4x \geq 8$$

$$2x \geq 3$$

$$x \geq 0$$

The solution is x = 2 , with 5x = 5×2 = 10 as expected .

The optimal solution to (A) is

$x = 2$, $y_1 = 1$, $y_2 = 0$ and $z^* = 10$.

2-3 PROBLEM FORMULATION

As reported earlier in the previous Chapter our main focus is on the Capacitated Spanning Tree problem (CST). We are given a graph G = (V,A) of n = |V| vertices and m = |A| arcs and requirements q_i, i = 2,...,n on each vertex except on vertex 1 that will be referred to as the central facility or root. We are also given Q_k quantities, k = 1,...,M and we are asked to find the clustering of the vertices of G

into M clusters in such a way that the total sum of the loads in any cluster k does not exceed Q_k and the spanning tree derived within these clusters has minimal cost among all those spanning trees induced by other clustering.

In this section we are going to discuss the use of Benders' decomposition scheme for a new integer programming formulation of the CST. In the past there have been several attempts to use this elegant technique to solve a broad range of combinatorial optimization problems. *FLORIAN et al [54]* have used the algorithm to schedule the movement of railway engines, *RICHARDSON [127]* has applied the algorithm to airline routing, and *GEOFFRION and GRAVES [73]* applied the algorithm to solve multi commodity distribution problems, *FISHER and JAIKUMAR [50]* discussed the advantages of using the algorithm for vehicle routing problems. These contributions and many others (*[57],[122]*) demonstrate the potential for using Benders' decomposition technique to solve structured mixed integer programs. However, not all of these applications were successful in applying Benders' decomposition because of the slow convergence of the algorithm. Geoffrion and Graves had the most success in using it efficiently. This brings us to ask ourselves why it is that this algorithm is so disappointing in practice except for very few cases. Geoffrion and Graves, among others, have noted that reformulating a mixed integer program can have a profound effect upon the efficiency of the algorithm. *WONG [141]* has observed that straightforward implementation of Benders' algorithm for a class of network design problems, often converged very slowly requiring the solution of an exorbitant number of integer problems. So it seems that selecting the proper model formulation for a given problem is an important factor that affects the computational performance of Benders' decomposition. This was best illustrated by *MAGNANTI and WONG [113]* when they used two equivalent integer formulations for the p-median facility location problem and showed that one formulation needed an exponential number of cuts to be derived in order for the algorithm to converge as compared for only one cut to be

derived for the second formulation for convergence. This shows the importance of shrewd model formulation for Benders' decomposition.

Bearing this in mind, we have tried to develop a new integer programming formulation for the CST that is well structured and lends itself to be used efficiently within the Benders framework.

The integer formulation for the Capacitated Spanning Tree problem (CST) is given below :

Let the set of $x_{\ell k}$ variables be such that :

$$x_{\ell k} = \begin{cases} 1 & \text{if arc } \ell \text{ belongs to the MST in subset k} \\ 0 & \text{otherwise} \end{cases}$$

Let the set of 0-1 y_{ik} variables be such that :

$$y_{ik} = \begin{cases} 1 & \text{if vertex i is in subset k} \\ 0 & \text{otherwise} \end{cases}$$

Then we can formulate the problem with the above sets of variables as :

$$\min \sum_{\ell=1}^{m} \sum_{k=1}^{M} c_{\ell} x_{\ell k} \qquad (1)$$

subject to :

$$\sum_{i=2}^{n} q_i y_{ik} \leq Q_k \quad , \quad k = 1,...,M \qquad (2)$$

$$\sum_{k=1}^{M} y_{ik} = 1 \quad , \quad i = 2,...,n \qquad (3)$$

$$\sum_{\ell \in A(S)} \sum_{k=1}^{M} x_{\ell k} \leq |S| - 1 \quad , \quad \begin{matrix} \forall\, S \subset N \\ 2 \leq |S| \leq n-1 \end{matrix} \tag{4}$$

$$\sum_{\ell=1}^{m} \sum_{k=1}^{M} x_{\ell k} = n - 1 \tag{5}$$

$$\left.\begin{matrix} x_{\ell k} \leq y_{i_\ell k} \\ x_{\ell k} \leq y_{j_\ell k} \end{matrix}\right\} \quad , \quad \begin{matrix} \ell = 1,\ldots,m \\ k = 1,\ldots,M \end{matrix} \tag{6}$$

$$x_{\ell k}\, ,\, y_{ik} \in \{0,1\} \quad , \quad \begin{matrix} \ell = 1,\ldots,m \\ k = 1,\ldots,M \end{matrix} \tag{7}$$

where A(s) : set of arcs having both ends in S ;

i_ℓ , j_ℓ are the extremities of arc ℓ ;

and m is the number of the arcs in the graph (if a complete graph then $m = n(n-1)/2$).

Constraints (2) and (3) are the capacity constraints. These constraints ensure that we have M different subsets (or clusters) of vertices, with vertex 1 included in each of these subsets (since vertex 1 does not have any load requirement on it, i.e. $q_1 = 0$), and that the total load in each subset does not exceed a given quantity Q_k. These constraints are those of a Generalized Assignment Problem (GAP). Constraints (2) ensure that for a given subset k the total sum of the requirements of the vertices belonging to that subset does not exceed Q_k. Constraints (3) force each vertex to belong to one and only one subset k.

Constraints (4) and (5) are one of the known constraints to mathematically model the definition of the spanning tree problem on a graph. Constraints (4) are the subtour elimination constraints. They ensure that the set of arcs chosen will contain no circuits and hence that they form a connected graph. Constraints (5) complete the

definition of a tree by making sure that the number of arcs chosen is exactly equal to (n-1) arcs for an n-vertex graph G. These two conditions reflect well the definition of a spanning tree (*CHRISTOFIDES [29]*).

Constraints (6) are the constraints that link the two sets of variables x_{ik} and y_{ik}. These make sure that an arc, if chosen to be part of the optimal solution, can not have its ends in two different subsets. Constraints (7) are the integrality constraints which means that we need the variables x_{ik} and y_{ik} to attain values of zero or one only.

Finally (1) represents the objective function of the program and indicates that we want to minimize the overall length of the tree chosen among all feasible trees that satisfy constraints (2) to (7).

As we see the formulation is based on two interconnected subproblems. The first one is a Generalized Assignment Problem (GAP) and the second is the Minimum Spanning Tree problem (MST). The formulation is attractive because of the fact that, first it characterizes the CST problem clearly as a tree problem with the additional capacity condition , and secondly this characterization is done through two combinatorial problems that have both been extensively studied. One of these problems, the MST problem, is one of the few combinatorial problems that are completely solved. As for the other problem (GAP), even though the problem is an NP-complete problem (*MARTELLO and TOTH [117]*), it has been extensively studied and some exact solutions have been derived for it.

2-4 OUTLINE OF THE ALGORITHM BASED ON BENDERS' DECOMPOSITION PROCEDURE

As we have seen in the previous section, the formulation of the CST consists of two interconnected subproblems, each defined on a set of 0-1 variables. This kind of formulation makes it appealing to use Benders' decomposition algorithm.

If we are given a set of feasible y_{ik} variables, i.e. satisfying constraints (2) and (3), then problem (P1) reduces to an integer program in the $x_{\ell k}$ variables only. When the y_{ik} are fixed, i.e. attaining values of 0 or 1, then the linking constraints (6) take the following form :

$$x_{\ell k} \leq 0 \tag{8}$$

$$x_{\ell k} \leq 1 \tag{9}$$

In view of constraints (7), constraints (8) can be tightened to

$$x_{\ell k} = 0 \tag{10}$$

and constraints (9) can be dropped because they are redundant.

Therefore, problem (P1) can be rewritten in the following manner :

$$\min \sum_{\ell=1}^{m} \sum_{k=1}^{M} c_{\ell k} x_{\ell k} \tag{11}$$

subject to :

$$\sum_{\ell \in A(S)} \sum_{k=1}^{M} x_{\ell k} = |S| - 1 \quad , \quad \forall S \subset V \quad 2 \leq |S| \leq n-1 \tag{12}$$

$$\sum_{\ell=1}^{m}\sum_{k=1}^{M} x_{\ell k} = n-1 \qquad (13)$$

$$x_{\ell k} = 0 \quad , \quad \ell=1,...,m \, , \quad k \in B \qquad (14)$$

$$x_{\ell k} \in \{0,1\} \quad , \quad \ell = 1,...,m \, , \quad k = 1,...,M \qquad (15)$$

where B is the set of indices ℓ and k such that $y_{i_\ell k} = 0$.

Program (11) to (13) is the mathematical formulation to the spanning tree problem plus additional constraints on the graph G = (V,A). Constraints (14) imply that some arcs ℓ cannot be part of the optimal solution. Therefore, one can use the mathematical programming formulation of the MST on a modified graph G* = (V*,A*) where the arcs ℓ induced by constraints (14) are deleted (or equivalently have their cost set to ∞). Thus, solving program (11)-(15) for graph G = (V,A) is equivalent to solving the following program on graph G* = (V*,A*) :

$$\min \sum_{\ell=1}^{m}\sum_{k=1}^{M} c_{\ell k} x_{\ell k} \qquad (16)$$

subject to :

$$\sum_{\ell \in A(S)}\sum_{k=1}^{M} x_{\ell k} = |S| - 1 \quad , \quad \forall S \subset V, \; 2 \le |S| \le n-1 \qquad (17)$$

$$\sum_{\ell=1}^{m}\sum_{k=1}^{M} x_{\ell k} = n-1 \qquad (18)$$

$$x_{\ell k} \in \{0,1\} \quad , \quad \ell=1,...,m \ , \ k=1,...,M \tag{19}$$

This formulation is naturally integer, meaning that dropping the integrality constraint on the $x_{\ell k}$ variables does not affect the final solution, always resulting in an integer solution when program (16)-(19) is solved as a linear program with constraint (19) ignored. In fact there is no need to use linear programming to solve program (16)-(19) since efficient greedy algorithms are available to solve it. Consequently, solving problem (11)-(15) for graph G = (V,A) reduces to transforming graph G = (V,A) into graph G* =(V*,A*) as explained above and then using Kruskal's or Prim's algorithm *[100,125]* to find the MST on graph G*. We will, from now on, be considering problem (16)-(19) as a linear program by replacing constraints (19) by :

$$x_{\ell k} \geq 0 \tag{20}$$

In the Benders partitioning procedure, problem (16)-(18) and (20) is the slave problem. We are in fact interested in getting the dual variables of problem (16)-(18) and (20). The dual program of (16)-(18) and (20) can be written as follows :

$$\max \sum_{S} \alpha_S(|S| - 1) + \beta(n - 1) + \overbrace{\sum_{i=2}^{n} \sum_{k=1}^{M} \gamma_{ik} \times 0}^{0} \tag{21}$$

subject to :

$$\sum_{S \supset \{i_\ell, j_\ell\}} \alpha_S + \beta + \gamma_{i_\ell k} \quad \leq \quad c_{\ell k} \tag{22}$$

$$\begin{aligned} &\alpha_S \leq 0 \\ &\beta \text{ unrestricted} \\ &\gamma_{ik} \leq 0 \end{aligned} \tag{23}$$

where i_ℓ , j_ℓ are the end vertices of arc ℓ.

To derive the Benders' cut one has to find an optimal solution to (21)-(23). This will be explained fully in the next Chapter.

Concerning the master problem, it will consist of the GAP constraints supplemented by the Benders' cuts and the Master program is as follows :

$$\text{Min } z \tag{24}$$

subject to :

$$\sum_{i=2}^{n} q_i y_{ik} \leq Q_k \quad , \quad k = 1,...,M \tag{25}$$

$$\sum_{k=1}^{M} y_{ik} = 1 \quad , \quad i = 2,...,n \tag{26}$$

$$z - \sum_{S} \alpha_S^p (|S| - 1) - \beta^p (n - 1) - \sum_{i=2}^{n} \sum_{k=1}^{M} \gamma_{ik}^p y_{ik} \geq 0 \quad , \quad p = 1,...,P \tag{27}$$

$$z \geq 0 \ , \ y_{ik} \in \{0,1\} \quad , \quad i=2,...,n \ , \quad k=1,...,M \tag{28}$$

One important feature of the application of Benders' procedure to our formulation is that we do not need to take care of the fact that we have to generate extreme rays of the feasible region defined by program (21)-(23). The reason is that whatever feasible solution we take for the GAP there will always be a feasible solution for the resulting linear program which is the MST defined on variables $x_{\ell k}$. The Benders partitioning procedure iterates between the two subproblems defined by (16)-(18) and (20) and (24)-(27), which we have to solve at each iteration optimally, each of the subproblems supplying a lower and an upper bound on the optimal value

of problem (1)-(7).

Now we are ready to formally state the complete algorithm of the Benders procedure as applied to the 0-1 integer programming formulation for the CST introduced in this Chapter.

STEP 1 Initialization.

Select a convergence tolerance parameter $\varepsilon \geq 0$.

Initialize UB = $+\infty$, LB = $-\infty$, H = 0.

If a binary array y^1 satisfying (2) and (3) is given, Goto Step3.

STEP 2 Solve the Master problem.

Solve the current master problem

min z

subject to :

(2), (3) and

$$z - \sum_S \alpha_S^h(|S| - 1) - \beta^h(n - 1) - \sum_i \sum_k \gamma_{ik}^h y_{ik} \geq 0 \qquad h = 1,\ldots,H$$

by any applicable algorithm.

Let (z^{h+1}, y^{h+1}) be any optimal solution.

Put LB equal to the optimal value of the above Master problem which is a lower bound on the optimal value of (1)-(7).

Terminate if UB $\leq$ LB + ε.

STEP 3 Solve the Slave problem.

(a) solve the linear programming problem :

constraint (1)

subject to :

(4),(5),(6) and (7) with y = y^{h+1}

by any applicable algorithm.

Denote the optimal value by $T(y^{h+1})$ and the optimal solution by x^{h+1}, then the quantity $T(y^{h+1})$ is an upper bound on the optimal value of (1)-(7).

If $T(y^{h+1})$ is less than UB replace UB by this quantity.

Store (x^{h+1}, y^{h+1}) as the incumbent and terminate if $UB \leq LB + \varepsilon$.

(b) Determine an optimal dual solution for (1) and (4)-(7) and denote it by α_s^h (corresponding to constraint (12)), β^h (corresponding to constraint (13)) and γ_{ik}^h (corresponding to constraint (14)).

Increase H by 1. Return to Step 2 .

The application of Benders partitioning procedure is straightforward for problem formulations that decompose naturally in two well defined subproblems, and this case is no exception. However, the problem that often arises in this kind of decomposition, is the fact that in two steps of the algorithm we need to solve to optimality two different problems, a task that is by no means easy, since one of these programs is an all 0-1 integer programming program. This is going to be one of the drawbacks of this procedure because long computational times will be required in order to derive an optimal solution to this program. As for the second problem being a linear program, linear programming packages are available to solve it. But the difficulty we face with this problem is the fact that we first need a dual optimal solution to it and secondly that there is an exponential number of constraints which limits severely the possibility to use LP packages to solve it. All of these difficulties and how to overcome them are explained in more detail in Chapter 3.

Nevertheless, it is interesting and worthwhile to look at problems from different angles and try to devise new formulations that may suit the kind of tools that we have in hand in order to solve them. The formulation presented in this Chapter is one of them.

2-5 A HEURISTIC FOR THE CST

Approximate solutions to combinatorial optimisation problems have always been an alternative to exact algorithms because their running time is polynomial, making them useful to solve large-sized problem even though we are not guaranteed that the solution is a "good" one, i.e. that the solution is not very far away from the optimum one. These heuristics are useful for those problems known to be NP hard.

In Chapter 1 we described and reported several heuristics for the CST. In our case we are studying a more general problem where the load capacity may vary from one subset to the other. The usual CST is the one where the load capacity for each subset is the same. This is where the need for a new heuristic arises for our more general case.

FISHER and JAIKUMAR [52] described a most interesting heuristic for the vehicle routing problem. Their vehicle routing problem was formulated as an integer program interconnecting two well known problems : a GAP problem and a traveling salesman problem (TSP). The similarities of the structure of their formulation with ours enabled us to adapt their heuristic to our problem.

The CST as formulated in (1)-(7) can be reformulated as follows :

$$\min \sum_{k=1}^{M} f(y_k) \tag{29}$$

subject to :

$$\sum_{i=2}^{n} q_i y_{ik} \leq Q_k \quad , \quad k = 1,...,M \tag{30}$$

$$\sum_{k=1}^{M} y_{ik} = 1 \quad , \qquad i = 2,...,n \qquad (31)$$

$$y_{ik} \in \{0,1\} \quad , \quad i = 2,...,n \; ; \quad k=1,...,M \qquad (32)$$

where $f(y_k)$ is the cost of an optimal minimum spanning tree in the subset :

$$N(y_k) = \{ i / y_{ik} = 1 \}$$

In fact the function $f(y_k)$ can be mathematically expressed but is extremely complicated and cannot even be written down for non trivial problems. As it is, program (29)-(32) is a non linear programming problem because of the non linearity of the objective function (29). The idea behind this formulation is to construct a linear approximation for $f(y_k)$ and then solve problem (29)-(32) with (32) replaced by :

$$\sum_{k=1}^{M} \sum_{i=2}^{n} d_{ik} y_{ik} \qquad (33)$$

Thus, problem (29)-(32) with (29) replaced by (33) is a Generalized Assignment Problem, and the solution for this GAP defines a feasible assignment of vertices to subsets (clusters). We then determine a feasible solution to the CST by solving an MST in each cluster of vertices. The problem, now, is how to approximate $f(y_k)$. We first begin with a set of M "seed" vertices $i_1,...,i_M$ that will be assigned one for each of the M subsets. Then we set the cost of assigning vertex i to subset k in the following manner :

$$d_{ik} = \min (c_{1i_k}, c_{ii_k})$$

In other words d_{ik} will be the shortest arc to choose between the arcs connecting the center to the "seed" vertex or the vertex i to the "seed" vertex (Figure 2-1).

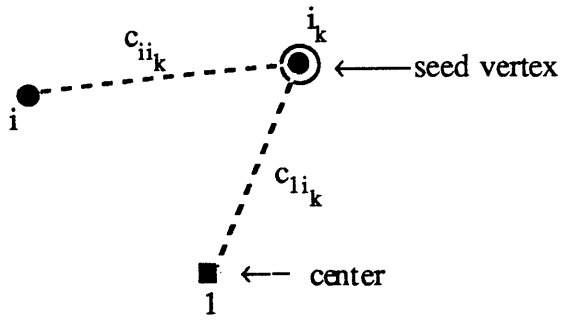


Figure 2.1

Then computing the d_{ik} for each the vertices i =2,...,n and each subset k =1,...,M gives a complete cost function to the GAP problem that can then be solved exactly by some available procedure (e.g. *MARTELLO and TOTH [117]*) for reasonable-sized problems whenever the capacity constraint is not very tight (i.e.$(\Sigma q_i/\Sigma Q_k) \longrightarrow 1$). The choice of "seed" vertices is the most important step in the heuristic because the quality of the solution will depend heavily on how the choices are made. The choice of "seed" vertices can made in several ways. The most interesting way would be for a person familiar with the problem to implement the algorithm on a computer system and interactively make different choices of "seed" vertices. Another way would be to have an automatic rule for the choice of the "seed" vertices. In our case the latter has been used to evaluate several test problems. The automatic rule was to take M vertices which are the furthest from the center and from each other and try to make several assignments for the choice of "seed" vertices among these M vertices. The following rule was used to derive the M seeds :

a) Pick furthest point as first seed.

b) Pick point for which the product of distance from centre and distance from all previously chosen seeds is maximum and call it the new seed.

c) Repeat (b) until M seeds are picked.

The results were very encouraging since on the average the solution was, on average, 10% away from the optimum T.

The following is an outline of the main procedure of the heuristic :

STEP 1 Selection of seeds.

Determine a set of M "seed" vertices $i_1,...,i_M$ following a given rule, that are assigned to the M subsets.

STEP 2 Computation of distances.

A coefficient d_{ik} is computed in the following manner :

$$d_{ik} = \min (c_{1i_k}, c_{ii_k})$$

where c_{1i} = distance of vertex i to center vertex 1.

c_{ii_k} = distance of vertex i to "seed" vertex i_k

also we set :

$d_{i_k k}$ = 0 (i.e. cost of assigning "seed" vertex i_k to subset k is 0).

STEP 3 Solution of the GAP.

Solve the following generalized assignment problem (GAP) :

$$\min \sum_{k=1}^{M} \sum_{i=2}^{n} d_{ik} y_{ik}$$

subject to :

$$\sum_{i=2}^{n} q_i y_{ik} \leq Q_k \quad , \quad k = 1,...,M$$

$$\sum_{k=1}^{M} y_{ik} \leq 1 \quad , \quad i = 2,...,n$$

$$y_{ik} \in \{0,1\} \quad , \quad i = 2,...,n ; \quad k = 1,...,M$$

The result will form M clusters of vertices. Denote them by $C_1,...,C_M$.

STEP 4 Solution of M spanning trees.

In each cluster C_k , k = 1,...,M ,solve the minimum spanning tree (MST), and denote its value by $V(C_k)$ then :

$$\sum_{k=1}^{M} V(C_k)$$

is an upper bound on the optimal value of the CST.

This procedure can be repeated several times with different choices of "seed" vertices and the best solution found so far will be taken as the best upper bound to the CST. There are, however, some limitations in repeating this procedure several times because of the drawback imposed by the exact solution for the GAP problem especially whenever the capacity constraint is tight (i.e. $(\Sigma q_i/\Sigma Q_k) \longrightarrow 1$). This limits the possibility of solving large sized problems because of the large amount of computing time that is needed to solve one single GAP. Of course one could also use a heuristic as a method of solution for the GAP, but one can not guarantee the behaviour of the final result of the CST because a heuristic embedded into another might lead to a solution far away from the optimum.

2-6 COMMENTS

Although Benders partitioning is an elegant method it has in fact worked for only very limited cases (e.g. *GEOFFRION and GRAVES [73]*). The underlying reasons are not well known but one factor which has a big effect on the behaviour of the procedure is problem formulation.

As it will be reported in Chapter 4, the new formulation presented in this Chapter enjoyed, in most cases, a fast convergence rate. The major drawback of the procedure was how to solve efficiently, in terms of computing time, the Master problem.

CHAPTER 3

SLAVE and MASTER PROBLEM SOLUTIONS

3-1 INTRODUCTION

In the previous Chapter a new 0-1 integer programming formulation for the CST was introduced and an elegant method to solve it was described. Benders' decomposition method requires the solution of two separate subproblems to be solved optimally for several iterations until a criterion of optimality stops the algorithm. These two subproblems which are denoted as a "Slave" problem and a "Master" problem are not easy problems to solve. The effectiveness of Benders' decomposition algorithm depends totally on the speed with which solutions can be derived from the Slave and Master problems. This is why so much attention is given to these two subproblems.

Section 3-2 introduces the solution of the Slave problem which consists mainly of

showing how to derive the dual variables of the Slave problem.

Section 3-3 involves the discussion and the proposed methods to use in order to solve the Master problem and as usual with combinatorial problems this section will introduce us to the development of a tree search for the solution of the Master problem.

3-2 COMPUTING THE DUAL VARIABLES

For the sake of convenience we will repeat below the details of the 0-1 programming formulation for the CST and its developments that were introduced in the last Chapter.

The problem is formulated in two sets of variables :

Let $x_{\ell k}$ and y_{ik} be defined as :

$$x_{\ell k} = \begin{cases} 1 & \text{if arc } \ell \text{ belongs to subtree } k \\ 0 & \text{otherwise} \end{cases}$$

$$y_{ik} = \begin{cases} 1 & \text{if vertex } i \text{ belongs to subtree } k \\ 0 & \text{otherwise} \end{cases}$$

Then we have the 0-1 integer program :

$$\min \sum_{\ell=1}^{m} \sum_{k=1}^{M} c_{\ell} x_{\ell k} \qquad (1)$$

subject to :

$$\sum_{i=2}^{n} q_i y_{ik} \leq Q_k \quad , \quad k = 1,...,M \qquad (2)$$

$$\sum_{k=1}^{M} y_{ik} = 1 \quad , \quad i = 2,...,n \qquad (3)$$

$$\sum_{\ell \in A(S)} \sum_{k=1}^{M} x_{\ell k} \leq |S| - 1 \quad , \quad \forall S \subset N, \; 2 \leq |S| - 1 \leq n - 1 \qquad (4)$$

$$\sum_{\ell=1}^{m} \sum_{k=1}^{M} x_{\ell \kappa} = n - 1 \qquad (5)$$

$$\begin{matrix} x_{\ell k} \leq y_{i_\ell k} \\ x_{\ell k} \leq y_{j_\ell k} \end{matrix} \quad , \quad \ell = 1,...,m \; ; \; k = 1,...,M \qquad (6)$$

$$x_{\ell k}, y_{ik} \in \{0,1\} \quad , \quad \ell = 1,...,m \; ; \; k = 1,...,M \qquad (7)$$

where n represents the number of vertices of the network, $m \triangleq \frac{1}{2}n(n-1)$, the number of arcs, M the number of subtrees we want to cluster the network in and c_ℓ the cost of having arc ℓ in any subtree k, and A(S) is the set of arcs having both terminal vertices in S. Also i_ℓ and j_ℓ are the terminal vertices of arc ℓ.

If we are given a set of feasible y_{ik} variables i.e. satisfying constraints (2)-(3) and (7) then problem (1)-(7) reduces to an integer (linear) program in the $x_{\ell k}$ variables only.

When we have a set of feasible 0-1 variables y_{ik} constraints (3-6) become as follows :

$$x_{\ell k} \leq 0 \tag{8}$$

or

$$x_{\ell k} \leq 1 \tag{9}$$

In view of constraints (7), then constraints (8) become :

$$x_{\ell k} = 0 \tag{10}$$

and constraints (9) can be dropped because they are redundant.

Thus, we are left with the following problem :

$$\min \sum_{\ell=1}^{m} \sum_{k=1}^{M} c_{\ell} x_{\ell k} \tag{11}$$

subject to :

$$\sum_{\ell \in A(S)} \sum_{k=1}^{M} x_{\ell k} \leq |S| - 1 \quad , \quad \forall\, S \subset N,\ 2 \leq |S| \leq n-1 \tag{12}$$

$$\sum_{\ell=1}^{m} \sum_{k=1}^{M} x_{\ell k} = n - 1 \tag{13}$$

$$x_{\ell k} = 0 \quad , \quad \text{for } \ell \in B ; \quad k = 1,...,M \tag{14}$$

$$x_{\ell k} \in \{0,1\} \quad , \quad \ell = 1,...,m; \quad k = 1,...,M \tag{15}$$

where B is the set of indices of ℓ such that $y_{i_{\ell} k} = 0$ and A(S) is the set of arcs having both terminal vertices in S

We have shown in Chapter 2 that problem (11)-(15) is the mathematical formulation for the MST plus additional constraints on graph G=(V,A) and it has been shown that its solution is equivalent to finding the MST on graph $G^*=(V^*,A^*)$ where the

arcs ℓ induced by constraints (15) are deleted from graph G (or equivalently have their cost set to ∞).

In Benders' decomposition terms problem (11)-(15) is the Slave problem. What we are in fact interested in is the derivation of an optimal dual solution to problem (11)-(14) and with constraints (15) replaced by :

$$x_{\ell k} \geq 0 \quad , \quad \ell = 1,...,m \; ; \; k = 1,...,M \tag{16}$$

As expressed above the mathematical formulation for the MST is of almost no use for deriving a solution to the problem because of the number of constraints that is involved. As one can see, constraints (12) grow exponentially with the number of vertices involved in graph G. Usually, one gets dual variables by solving the corresponding linear program by means of linear programming techniques. Obviously this is not possible in this case with formulation (11)-(15).

The dual problem of (11)-(15) in terms of the dual variables α_S, β_S and γ_{ik} is written as follows :

$$\max \sum_S \alpha_S(|S| - 1) + \beta(n-1) + \overbrace{\sum_{i=2}^{n} \sum_{k=1}^{M} \gamma_{ik} \times 0}^{0} \tag{17}$$

subject to :

$$\sum_{S \supset \{i_\ell, j_\ell\}} \alpha_S + \beta + \gamma_{i_\ell} \leq c_\ell \quad , \quad l = 1,...,M \tag{18}$$

$$\left.\begin{array}{l} \alpha_S \leq 0 \\ \beta \text{ unrestricted} \\ \gamma_{ik} \leq 0 \end{array}\right\} \tag{19}$$

where i_ℓ and j_ℓ are the terminal vertices of arc ℓ.

One important feature of this dual program is that the variables γ_{ik} do not play any role in the objective function of the dual program. Since from duality theory we have:

$$\max \sum_{S} \alpha_S(|S| - 1) + \beta(n - 1) = \min \sum_{\ell=1}^{m} \sum_{k=1}^{M} c_{\ell k} x_{\ell k} \qquad (20)$$

we can say that the dual variables α_S and β defined on problem (11)-(15) for graph G=(V,A) will be the same dual variables defined on problem (11)-(16) for graph $G^*=(V^*,A^*)$.

This is an important step in getting a dual optimal solution (α_S , β , γ_{ik}) for problem (11)-(15) by an indirect method. This involves, first the derivation of a dual optimal solution (α_S , β) for an MST defined on graph $G^*=(V^*,A^*)$ and then get γ_{ik} from constraint (18).

The reason behind all this, is that it is possible to derive a dual optimal solution for the MST without having to make use of any linear programming package, but instead use a much faster, specialized, polynomially bounded algorithm.

3-2-1 Edmonds' algorithm

If we formulate the MST as a linear programming problem and want to get the dual variables using linear programming, we would not be able to solve problems involving more than 10 vertices because of the exponential number of constraints in that formulation, rendering any LP package worthless.

Therefore, another direction is needed to avoid the drawbacks of a formulation with an exponential number of constraints . An algorithm was originally devised by *EDMONDS [44]* and independently by *BOCK [20]*, which solves the more difficult case of the directed minimum spanning tree rooted at a given vertex v_0 in a

directed graph G=(V,A).

To use this algorithm for undirected graphs, we just add an artificial vertex v_0 and connect v_0 with every vertex $i \in V$ and set the cost for each arc (v_0,i) to a large number L_i. Also, for any directed arc (i,j) another directed arc (j,i) is added with the same cost as arc (i,j). The resulting graph G^0 is then used to derive the minimum directed spanning tree rooted at v_0.

Edmonds' Algorithm :

STEP 1 Initialization.

k=0 i.e. $G^k = G^0 = G$.

STEP 2 Selection of least cost arcs.

For each vertex(or pseudovertex) v_i (except for the root v_0) in the actual graph G^k, select the arc with terminal vertex v_i having least cost (see Figure 3-1).

STEP 3 Shrinking process.

If the addition of an arc to the solution produces a circuit ,this circuit is identified and shrunk into a single pseudovertex (the original vertices of the circuit are replaced by one vertex).

We will call G^k the graph generated after the k^{th} shrinking, and v_{n+k} the k^{th} generated pseudovertex (see Figure 3-1 and Figure 3-2).

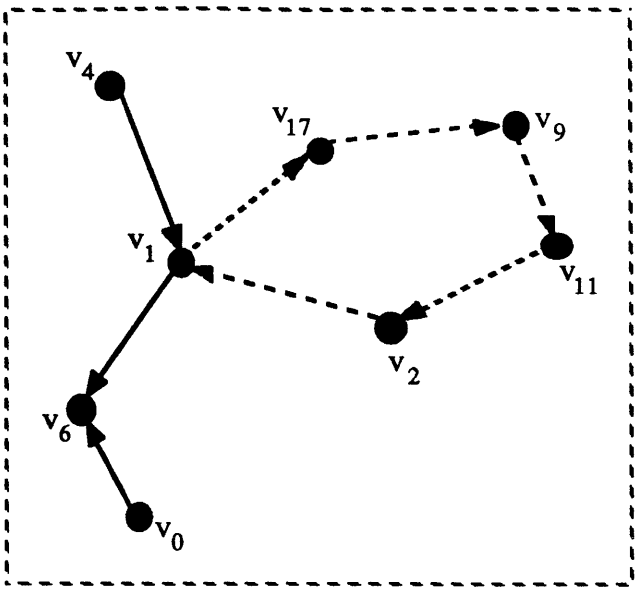


Figure 3-1 : Graph G^{k-1} where a circuit is detected in the first iteration of the algorithm.

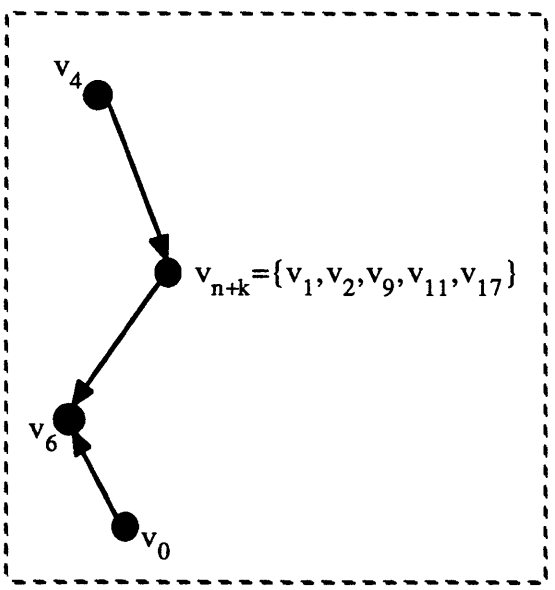


Figure 3-2 : Graph G^k after the circuit detected in G^{k-1} has been shrunk.

Let T_{n+k} be the set of vertices of graph G^{k-1}, which represents the previous graph before the k^{th} -1 shrinking, in the circuit to be shrunk. Then the costs of the adjacent arcs for v_{n+k} are :

for arc (v_j, v_{n+k}) $$c_{j,n+k} = \min_{v_t \in T_{n+k}} \{c_{jt} - c_{a(t)t}\} \quad (21)$$

for arc (v_{n+k}, v_j) $$c_{n+k} = \min_{v_t \in T_{n+k}} c_{tj} \quad v_j \in G^{k-1} \quad v_j \notin T_{n+k} \quad (22)$$

for all j=0,...,n+k-1 $v_j \in G^{k-1}$ $v_j \notin T_{n+k}$

where a(t) means the predecessor vertex of v_t in the circuit.

STEP 4 Stopping rule.

If the indegree of all vertices (except for the root v_0) is 1 then

GOTO STEP5.

Otherwise k = k + 1 and GOTO STEP2.

STEP 5 Blossoming and extraction of the solution.

The circuits that have been shrunk are expanded in reverse order. Let v_{n+k} be expanded. v_{n+k} has an entering arc in the solution say (v_j, v_{n+k}), then we identify the vertex $v_t \in T_{n+k}$ (T_{n+k} being the set of vertices forming the circuit detected in graph G^{k-1}),that made minimum (20) in STEP3 which corresponds to a real arc of G^{k-1}, that is (v_j, v_t) , then we eliminate of the solution the arc $(v_{a(t)}, v_t)$.

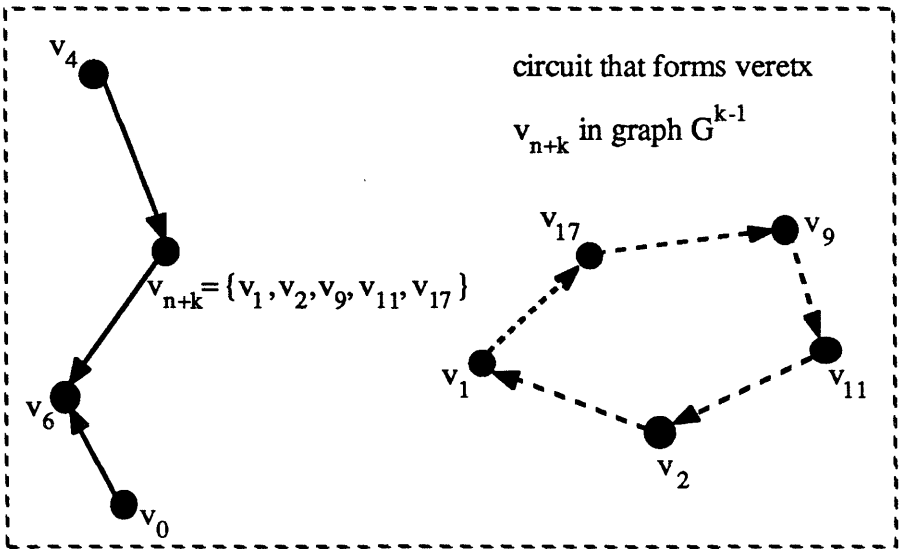


Figure 3-3 : Blossoming process.

The arc (v_4,v_{n+k}) is the entering arc in the solution in graph G^k in the left of the figure. This arc corresponds to the arc (v_4,v_1) in graph G^{k-1}.

The arc (v_1,v_2) in the circuit corresponds to the arc $(v_{a(t)},v_t)$ which must be eliminated from the solution.

Example :

To show the application of Edmonds' algorithm, we consider the following small example of how the algorithm works in practice.

The following matrix is the cost matrix of a 6 vertex directed graph G, with vertex 1 being the root.

$C =$

	1	2	3	4	5	6
1	-	8	12	-	-	4
2	-	-	10	-	-	-
3	-	-	-	-	9	-
4	-	7	1	-	-	-
5	-	-	-	9	-	-
6	-	-	-	25	20	-

Let us now apply Edmonds' Algorithm .

1st iteration :

STEP 1 Initialization of the algorithm.

The above cost matrix yields the following graph representation of $k = 0$; $G^k = G^0 = G$.

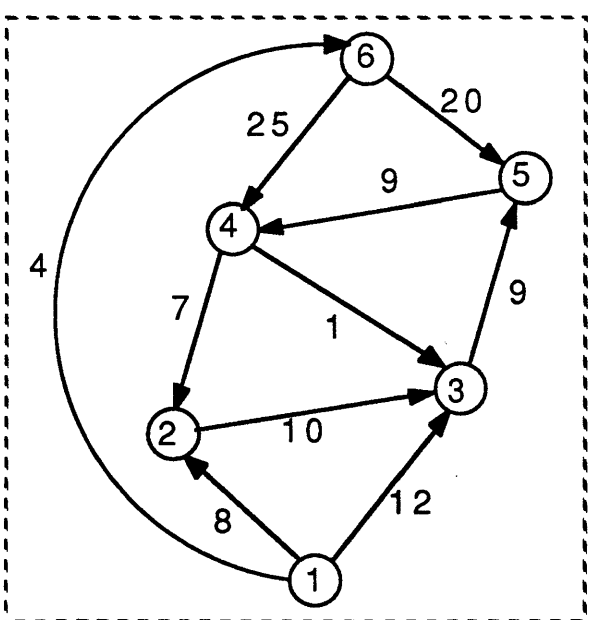


Figure 3-4 : Original graph G before the algorithm starts.

STEP 2 Selection of least cost arcs.

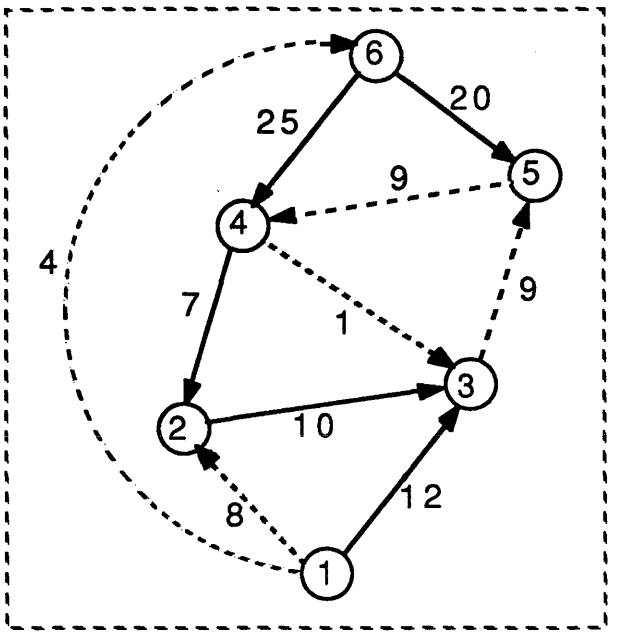


Figure 3-5 : Graph G^0 with the arcs selected to enter the solution. Vertex 1 is the root. Dotted lines represent the arcs selected to enter the solution.

STEP 3 Shrinking process.

Vertices 3,4 and 5 form a circuit in graph G^0. We contract it and get network G^1, updating the arcs and their cost accordingly.

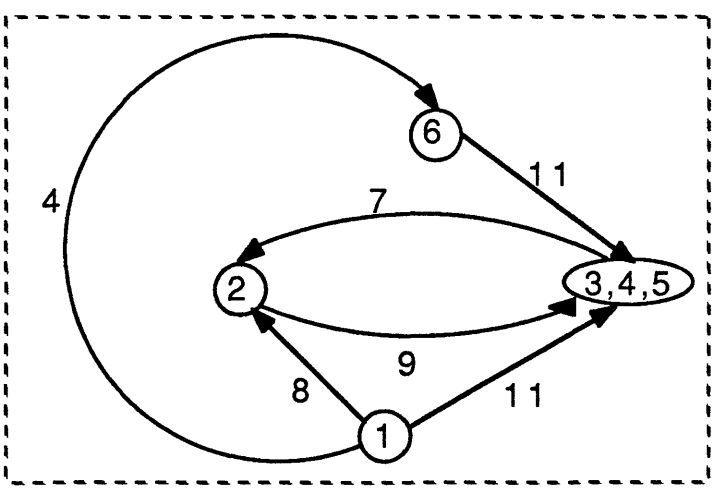


Figure 3-6 : Graph G^1 after 1st shrinking

STEP 4 Stopping rule.

In graph G^1 not all the vertices (except for 1) have indegree 1 so the algorithm iterates again with k = k +1 = 1. GOTO STEP 2.

2nd iteration :

STEP 2 Selection of least cost arcs.

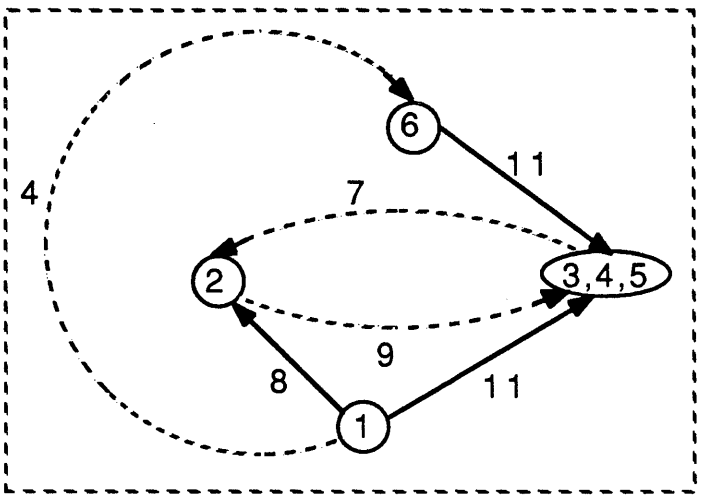


Figure 3-7 : Graph G^1 after the selection of the arc with least cost. Dotted lines represent the arcs selected.

STEP 3 Shrinking process.

Vertices {3,4,5} and 2 form a circuit in graph G^1. We contract it and get another graph G^2, updating the arcs and their costs accordingly.

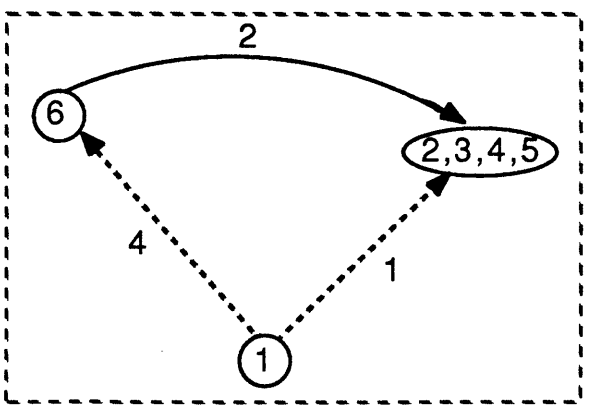


Figure 3-8 : Graph G^2 after 2nd shrinking.

STEP 4 Stopping rule.

In graph G^2 all the vertices (except vertex 1 the root) have their indegree equal to 1. Therefore, the algorithm terminates and what is left now, is to collect the arcs that will form the maximum branching. This is done in STEP 5.

STEP 5 Blossoming and extraction of the solution.

The blossoming is done in the reverse order to the one which created the shrinking. In the process the arcs forming the optimal solution to the maximum branching are collected.

1st blossoming :

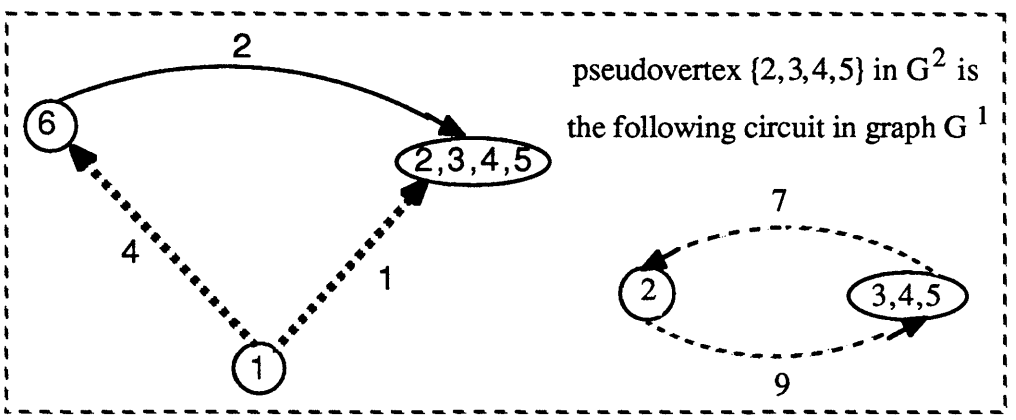


Figure 3-9 : Graph G^2 and the circuit in G^1 that was shrunk to become pseudovertex {2,3,4,5}.

We collect arc (1,6) to be in the final solution. The pseudovertex {2,3,4,5} in graph G^2 has the arc (1,{2,3,4,5}) as the entering arc in the solution. Arc (1,{2,3,4,5}) in graph G^2 corresponds to arc (1,2) in graph G^1, then arc (1,2) is entered in the final solution and arc ({3,4,5},2) is eliminated.

2nd blossoming :

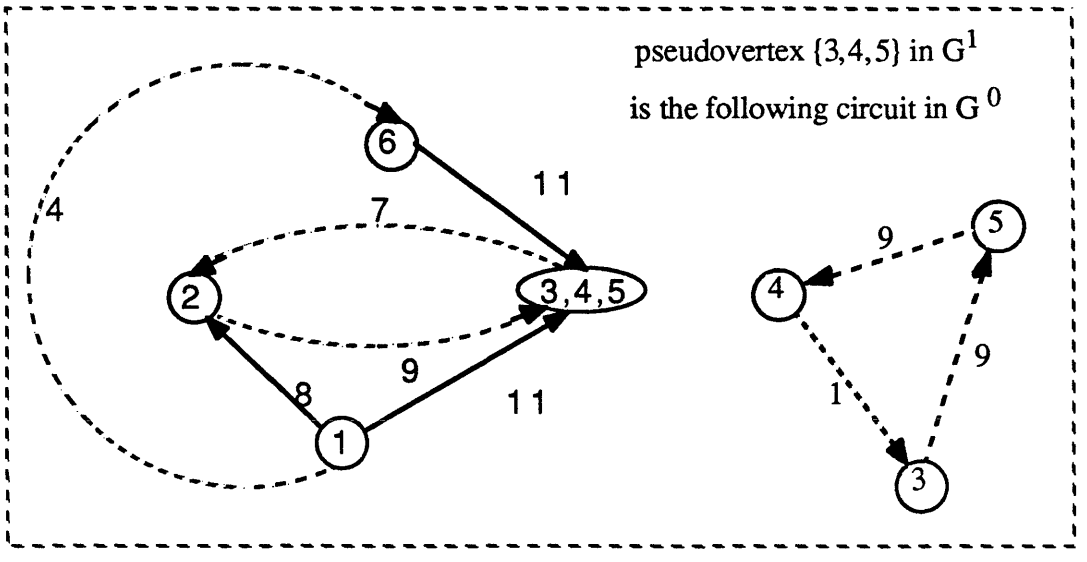


Figure 3-10 : Graph G^1 and the circuit in G^0 that was shrunk to become pseudovertex {3,4,5}.

The pseudovertex {3,4,5} has the arc (2,{3,4,5}) in graph G^1 as the entering arc in the solution. This arc corresponds to the arc (2,3) in graph G^0.This arc is selected to be in the final solution and we eliminate the arc (4,1) of the optimal solution.

The process of blossoming is complete and thus far, the arcs selected to be in the final optimum solution are (1,6), (1,2), (2,3) and in the first step from the first iteration we should also consider arcs (3,5) and (5,4) to be part of the optimal solution. Finally the optimal solution is pictured below :

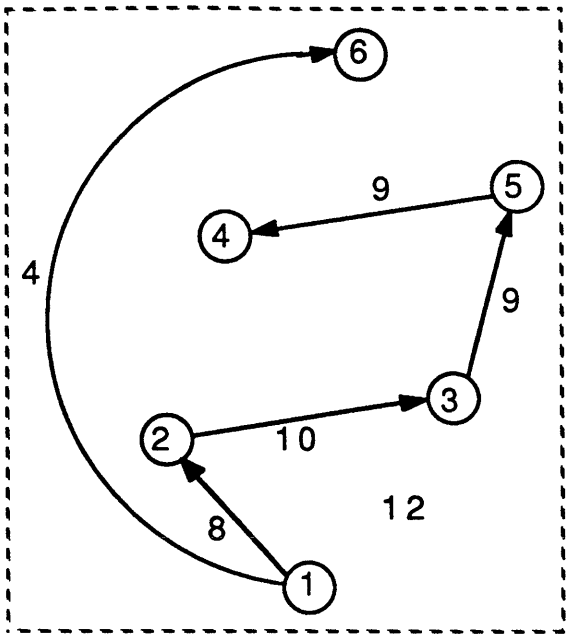


Figure 3-11 : Optimal Solution.

The cost of the above solution is 40.

3-2-2 Deriving the dual variables

The algorithm described in the last section yields a minimum directed spanning tree when there is a specified vertex given as a root. When the graph G=(V,A) is not directed , the only modification to make is to add an artificial vertex v_0 with additional arcs added from v_0 to each vertex $i \in V$ and set the costs for arcs (v_0,v_i) to a large number L_i. Also, for any directed arc (i,j) another directed arc (j,i) is added with the same cost as arc (i,j). The resulting graph G^* is then used to derive the minimum directed spanning tree. Only one of the arcs added joining the artificial root vertex to the other vertices will be chosen in the final solution because of their high cost. Edmonds' algorithm with a few modifications can yield at the same time the dual variables corresponding to the minimum directed spanning tree and also the dual variables for the case of the undirected minimum spanning tree (by means of the transformations described above).

Let us consider the mathematical programming formulation for the directed minimum spanning tree and its mathematical dual :

(Primal) :

$$\min \sum_{(i,j)} c_{ij} x_{ij} + \sum_{i=1}^{n} L_i x_{0i} \tag{23}$$

subject to :

$$\sum_{i=1}^{n} x_{ij} + x_{0j} = 1 \quad , \quad j = 1,...,n \tag{24}$$

$$\sum_{i,j \in S} -(x_{ij} + x_{ji}) \geq 1 - |S| \quad , \quad \forall\, S \subset V \tag{25}$$

$$x_{ij}, x_{0i} \geq 0 \quad , \quad i = 1,...,n \; ; \; j = 1,...,n \tag{26}$$

(Dual) :

$$\max \left\{ \sum_{i=1}^{n} \mu_i + \sum_{S} \gamma_S (1 - |S|) \right\} \tag{27}$$

subject to :

$$\mu_j - \sum_{S \supset \{i,j\}} \gamma_S \leq c_{ij} \quad , \quad \forall\, (i,j) \in A \tag{28}$$

$$\mu_i \leq L_i \quad , \quad i = 0,...,n \tag{29}$$

$$\gamma_S > 0 \, , \, \mu_i \in R \quad , \quad \forall\, S \subset V \; ; \; i = 0,...,n \tag{30}$$

The orthogonality conditions give the following :

$$\gamma_S\left[\sum_{(i,j)} -(x_{ij}+x_{ji})+|S|-1\right]=0 \qquad , \qquad \forall\, S \subset V \qquad (31)$$

and

$$x_{ij}\left[\mu_j - \sum_{S\supset\{i,j\}} \gamma_S - C_{ij}\right]=0 \qquad , \qquad \forall\, (i,j) \in A \qquad (32)$$

$$x_{0i}[\mu_i - L_i] = 0 \qquad , \qquad i = 0,...,n \qquad (33)$$

Given a solution [x] for problem (**Primal**), then we can describe a procedure that will lead us to the optimal solution for problem (**Dual**). The procedure can be implemented in Edmonds' algorithm and computed during the process of finding an optimal solution to the directed minimum spanning tree.

γ_S variables :

Let us recall that in Step 3 of Edmonds' algorithm a pseudovertex v_{n+k} is created when a circuit is formed. Let us call S_k the total set of vertices of graph G included in the pseudovertex v_{n+k}. In S_k the individual vertices included in any pseudovertex are also included and so on.

For all those sets S_k created in Step 3 of Edmonds' algorithm it is easy to see that the second factor of (31) will be 0 i.e :

$$\sum_{\{i,j\}\in S_k} -(x_{ij}+x_{ji})+|S|-1=0 \qquad (34)$$

Thus, we will allow for those sets S_k that $\gamma_S \neq 0$ with equation (30) still holding.

Let us take for those $\gamma_S \neq 0$ to be :

$$\gamma_S = c_{i,n+k} \qquad (35)$$

where (v_i,v_{n+k}) was the arc selected by the algorithm in Step 2 to be the entering arc for v_{n+k}.

For all the other γ_S we can set their value to 0.

μ_i variables :

$$\text{If } x_{0i} = 1 \text{ then } \mu_i = L_i \tag{36}$$

$$\text{If } x_{ij} = 1 \text{ then } \mu_j = c_{ij} + \sum_{S \supset \{i,j\}} \gamma_S \tag{37}$$

With these choices all the orthogonality conditions are verified.

What is now left to complete the construction of the dual variables (γ,μ), is to show that these dual variables are feasible for the dual problem (**Dual**) and that it has the same cost as [x].

Theorem 1: The solution (γ,μ) as defined in (35) , (36) and (37) is feasible for problem (**Dual**).

Proof [1] :

we have to show that

$$\mu_j - \sum_{S \supset \{i,j\}} \gamma_S \leq c_{ij} \quad , \quad \forall (i,j) \tag{38}$$

and

$$\mu_i \leq M_i \quad , \quad \forall i \in N \tag{39}$$

1st equation (38) :

We have to prove that (38) is satisfied for all arcs (v_i,v_j) $j = 1,...,n$

We distinguish two cases :

Case 1 : v_j is not the root.

This means that $x_{0j} = 0$ which implies that there is an $i \neq 0$ for which $x_{ij} = 1$.

Given that μ_j is chosen to be equal to :

$$c_{ij} - \sum_{S \supset \{i,j\}} \gamma_S \tag{40}$$

leads to the following equation by substituting (40) into (38) :

[1] An Alternative proof of this therorem is given by *JOHNSON E. [82]*

$$c_{ij} - \sum_{S \supset \{i,j\}} \gamma_S + \sum_{S \supset \{i,j\}} \gamma_S = c_{ij} \tag{41}$$

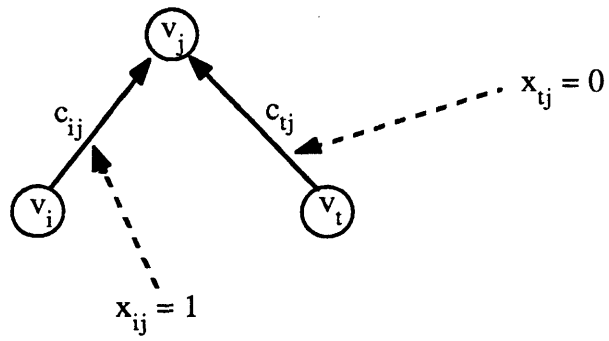
Therefore (38) is satisfied.

For another arc (v_t,v_j) , $t \neq i$ we have to show that :

$$c_{ij} + \sum_{S \supset \{i,j\}} \gamma_S - \sum_{S \supset \{t,j\}} \gamma_S \leq c_{ij} \Leftrightarrow c_{ij} + \sum_{S \supset \{i,j\}} \gamma_S \leq c_{tj} + \sum_{S \supset \{t,j\}} \gamma_S \tag{42}$$

(a) Let us suppose that arcs (v_i,v_j) and (v_t,v_j) are such that there is no S_k such that $\{i,j\} \subset S_k$ or $\{t,j\} \subset S_k$.

Then we have $c_{ij} \leq c_{tj}$ because of the fact that arc (v_i,v_j) is optimal (i.e. in the optimal solution) so the algorithm will have chosen arc (v_i,v_j) to be the entering arc for vertex v_j.



Obviously vertex v_j may itself be part of one or more pseudovertices but the transformations on the arcs (v_i,v_j) and (v_t,v_j) according to (20) and (21) in Edmonds' algorithm will be the same for both arcs.

Let us call c'_{ij} and c'_{tj} the arcs costs after these transformations then :

$$c'_{ij} \leq c'_{tj} \Leftrightarrow c_{ij} \leq c_{tj} \tag{43}$$

(b) Let us now suppose that S_r is the first set for which $\{i,j\} \subset S_r$ or $\{t,j\} \subset S_r$. It may be that vertex v_j had been part of another pseudovertex but the transformations up to now of all arcs entering v_j have been transformed according to (a).

(b1) Let arc $\{i,j\}$ be included in S_r first.

Let H_r be the set of arcs that were produced by the circuit made of the vertices in S_r.

It is clear that $(v_i,v_j) \in H_r$ because the arc would have been removed during the process of the expansion in Edmonds' algorithm and therefore would not have ended up in the optimal solution.

It may be possible that $\{t,j\} \subset S_r$ but then arc $(t,j) \notin H_r$ because $x_{tj} = 0$ (hypothesis) and also $c_{ij} \leq c_{tj}$ because of (a) and from that moment both $\{i,j\}$ and $\{t,j\}$ are included in S_k and then (38) is satisfied.

Let us now suppose that $\{t,j\}$ is not included in S_r. Arc (v_t,v_j) cannot be in the optimal solution because otherwise arc (v_i,v_j) would not be selected to be the entering arc to v_j.

If γ_{S_r} is the cost of the arc selected to enter the pseudovertex v_{n+r} then we have:

$$\gamma_{S_r} \leq c_{tj} - c_{ij}$$

where c_{tj} - c_{ij} is the modified cost of arc (v_i,v_j) because of the circuit S_r.

Written differently we have :

$$c_{ij} + \gamma_{S_r} \leq c_{tj}$$

If the pseudovertex v_{n+r} appears afterwards in another pseudovertex v_{n+r_1}, the cost of arc (v_t,v_j) will be modified by $-\gamma_{S_{r_1}}$. Then the arc selected to enter pseudovertex v_{n+r_1} will have cost $\gamma_{S_{r_1}}$ such that :

$$\gamma_{S_{r_1}} \leq c_{tj} - c_{ij} - \gamma_{S_r} \Rightarrow c_{ij} + \gamma_{S_r} + \gamma_{S_{r_1}} \leq c_{tj}$$

And iteratively if S_{r_p} is the last set that contains $\{i,j\}$ and not $\{t,j\}$ and if $\gamma_{S_{r_p}}$ is the cost of the arc entering the pseudovertex consisting of the set S_{r_p} then :

$$\gamma_{S_{r_p}} \leq c_{tj} - c_{ij} - \sum_{k=0}^{P-1} \gamma_{S_k} \qquad \text{where } S_{r_0} = S_r$$

the above implies then :

$$c_{ij} + \sum_{k=0}^{P-1} \gamma_{S_{r_k}} \leq c_{tj}$$

Therefore (38) is satisfied.

(b2) Let us suppose now that $\{t,j\} \subset S_r$.

Two cases appear :

1) If $\{i,j\} \not\subset S_r$ then arc $(v_i,v_j) \in H_r$ and we have $c_{ij} \leq c_{tj}$ and from now on both $\{i,j\}$ and $\{t,j\}$ are included in the set S_r, thus (38) is satisfied.

2) If $\{i,j\} \subset S_r$ we have either arc (v_t,v_j) belongs to H_r or not. If (v_t,v_j) does not belong to H_r then another arc (v_ℓ,v_j) belongs to H_r with $c_{\ell j} \leq c_{tj}$ but from now on (v_ℓ,v_j) and (v_t,v_j) belong to the same set S_r then it is sufficient to prove (38) for arc $(v_\ell,v_j) \in H_r$.

If $(v_t,v_j) \in H_r$ then let $S_r, S_{r_1}, \ldots, S_{r_p}$ be the sets that contain arc (v_t,v_j). Arc (v_i,v_j) must appear as the entering arc for at least one of those pseudovertices corresponding to sets S_r, S_{r_1}, ..., S_{r_p} and this because arc (v_i,v_j) appears in the final solution. Let this pseudovertex correspond to set S_{r_p} then :

$$\gamma_{S_{r_p}} = c_{ij} - c_{tj} - \sum_{k=0}^{P-1} \gamma_{S_{r_k}}$$

where the second member of the above equation is the modified cost of arc (v_i,v_j).

Then we have :

$$c_{ij} = c_{tj} + \sum_{k=1}^{P} \gamma_{S_{r_k}}$$

And from this point on both $\{i,j\}$ and $\{t,j\}$ are included in the same set and thus (38) is satisfied.

Case 2: v_j is the root.

If v_j is the root that is $x_{0j}=1$ then $\mu_j = L_j$ which is the cost of the artificial arc (v_0,v_j). It is clear that this arc can not belong to any circuit so we have to show that :

$$L_j \le c_{tj} + \sum_{S \supset \{t,j\}} \gamma_S \qquad \forall\, t$$

and the proof will be analogous to Case 1 for (a) and (b)

2nd equation:

We want to show that :

$$\mu_j \le L_j \qquad j = 0,1,\ldots,n$$

Case1 : v_j is the root.

It is obvious because $\mu_j = L_j$.

Case 2 : v_j is not the root.

We must show that :

$$\mu_j = c_{ij} + \sum_{S \supset \{i,j\}} \gamma_S \le L_j \longrightarrow \text{where } L_j \text{ is the cost of } (v_0,v_j)$$

The proof is similar to Case1 in the 1st equation proof.

Theorem 2 : The solutions (x_{ij}) for (**Primal**) and (γ_S,μ_i) for (**Dual**) have the same cost.

Proof:

Let us simplify the notation.

Whenever $x_{ij} = 1$ then we take $j = p(i)$ meaning that j is the predecessor of i in the final tree graph solution. Therefore, we have to show that :

$$\sum_{i \ne j} c_{p(i)i} + L_j = \sum_{i=1}^{n} \mu_i + \sum_{S} \gamma_S (1 - |S|)$$

where j is the index corresponding to the root of the solution after removal of the artificial root 0. We recall that $x_{0j} = 1$ implies that $\mu_j = L_j$.

Therefore, we could simplify the above equation by removing L_j from each side i.e. it can be rewritten as :

$$\sum_{i \neq j} c_{p(i)i} = \sum_{i \neq j} \mu_i + \sum_{S} \gamma_S (1 - |S|)$$

Furthermore we have that for any $i \neq j$:

$$\mu_i = c_{p(i)i} + \sum_{S \supset \{p(i),i\}} \gamma_S$$

For any $\gamma_S \neq 0$ it is clear that the number of arcs in the final solution is $|S| - 1$. Then each $\gamma_S \neq 0$ appears exactly in $(|S| - 1)$ μ_i's.

Thus :

$$\sum_{i \neq j} \mu_i = \sum_{i \neq j} c_{p(i)i} + \sum_{i \neq j} \sum_{S \supset \{i,j\}} \gamma_S$$

$$= \sum_{i \neq j} c_{p(i)i} + \sum_{S} \gamma_S (|S| - 1)$$

Thus, we have constructed a dual solution for the problem of finding a directed (or undirected) minimum spanning tree which can be computed without using a standard linear programming package. Edmonds' algorithm is known to be polynomially bounded (*LAWLER [100]*) and of complexity order $O(n^2 \log n)$.

Example :

Let us take the graph in the previous example and derive the dual variables for the corresponding directed minimum spanning tree.

Applying Edmonds' algorithm to the above graph where vertex 1 is specified as the root we have a first circuit consisting of vertices {3,4,5}

After contraction and updating of the arcs of the new graph we take the value of

$\gamma_{S=\{3,4,5\}}$ to be the value of the arc entering the pseudovertex consisting of vertices 3,4 and 5 in Step 2 of Edmonds' algorithm; that is :

$$\gamma_{S=\{3,4,5\}} = 9$$

A new circuit is created after the latter step and vertex 2 and pseudovertex {3,4,5} are collapsed into a new pseudovertex consisting of vertices {2,3,4,5}.

A new graph G^2 is created where the value of the entering arc for pseudovertex {2,3,4,5} is of value

$$\gamma_{S=\{2,3,4,5\}} = 1$$

The algorithm stops after only two iterations and an optimal solution to the directed minimum spanning tree is constructed in the reverse order to the one in which the pseudovertices that have been created (blossoming process).

We, therefore, set all other possible γ_S to value 0.

Now we can get the μ_i dual variables by using the formula :

$$\mu_i = c_{ji} + \sum_{S \supset \{i,j\}} \gamma_S \quad \text{for that arc (j,i) which is in the solution.}$$

By definition $\mu_1 = 0$ and :

$$\mu_2 = c_{12} + \sum_{S \supset \{1,2\}} \gamma_S$$
$$= c_{12} = 8$$

$$\mu_3 = c_{23} + \sum_{S \supset \{2,3\}} \gamma_S$$
$$= 10 + 1 = 11$$

$$\mu_4 = c_{54} + \sum_{S \supset \{4,5\}} \gamma_S$$
$$= 9 + 9 + 1 = 19$$

$$\mu_5 = c_{35} + \sum_{S \supset \{3,5\}} \gamma_S$$
$$= 9 + 9 + 1 = 19$$

$$\mu_6 = c_{16} + \sum_{S \supset \{1,6\}} \gamma_S$$

$$= 4$$

The objective of the dual is :

$$\sum_i \mu_i + \sum_S \gamma_S (1 - |S|)$$

Therefore, we get :

Value of (**Dual**) = (0+18+11+19+19+4) + (-2×9 + -3×1)

= 61 - 21 = 40 which is the same value as that of (**Primal**)

This example shows how straightforward and natural it is to derive the dual variables of the directed (or undirected) minimum spanning tree problem.

Also, we note that our formulation for the case of the undirected minimum spanning tree, the graph is easily transformed so that we can apply Edmonds' algorithm. Another point is that for the latter case, the algorithm will be used on a graph that will have all the arcs directed and in this way constraint :

$$\sum_i \sum_j x_{ij} = n - 1$$

is a surrogate constraint for:

$$\sum_i x_{ij} = 1 \qquad \text{for } j = 2,\ldots,n$$

where vertex 1 is considered as the root vertex.

Contrary to the case of the directed graph (see for example the previous numerical example) all cases of non-directed graphs that have been solved, we have obtained dual variables μ_i that are equal except for the root and, therefore, we have taken the dual variable β corresponding to constraint :

$$\sum_i \sum_j x_{ij} = n - 1$$

as being equal to any of the μ_i (except for the root vertex) .

The reason is that the way the directed graph is derived from the non-directed one

makes it so that every vertex has exactly the same number of arcs entering as there are ones leaving it. This fact forces all the μ_i dual variables to be equal (except for the root).

3-3 MASTER PROBLEM SOLUTIONS

As reported earlier in the previous Chapter, although Benders' decomposition is an elegant and seemingly efficient procedure, it could be useless if one of the two subproblems (Master or Slave) cannot be solved to optimality. In the previous section we have shown that the Slave problem can be solved optimally in very reasonable computing time since Edmonds' algorithm is polynomially bounded (*LAWLER [100]*).

The Master problem is more difficult since it is an integer programming problem.

In fact a successful implementation of Benders' procedure rests with an efficient solution to the Master problem. Also, the number of iterations required to reach optimality in Benders' procedure is of great importance.

In our specific case as seen in Chapter 2 the Master problem is formulated as follows :

$$\min\ z \tag{44}$$

subject to :

$$\sum_{k=1}^{M} y_{ik} = 1 \quad , \quad i = 2,...,n \tag{45}$$

$$\sum_{i=2}^{n} q_i y_{ik} \leq Q_k \quad , \quad k = 1,...M \tag{46}$$

$$z - \left(\sum_{S} \alpha_S^p (|S| - 1) + \beta^p (n - 1) \right) - \sum_{i} \sum_{k} \gamma_{ik}^p y_{ik} \geq 0 \quad , \quad p = 1,...,P \tag{47}$$

$$z \geq 0 \quad , \quad y_{ik} \in \{0,1\} \quad , \quad i = 2,...,n \; ; \; k = 1,...,M \tag{48}$$

The above problem is well structured when P = 0 which reduces it to the familiar General Assignment Problem (GAP). In fact when the Benders' procedure is initiated, a feasible solution to constraints (45)-(48) is sought in order to start the algorithm. But as more cuts are added the structure of the GAP is destroyed making the problem (44)-(48) more difficult to solve.

One useful advantage of the Benders' decomposition procedure is that it leaves us the freedom to choose whatever appropriate method is needed to solve the master problem i.e. problem (44)-(48).

3-3-1 Cutting plane algorithms for the Master problem

Not many choices are available when one is faced with the prospect of solving an integer program. In fact two main techniques have been extensively used to tackle integer programming problems : Cutting-plane algorithms and tree search methods.

Since the first paper on the theory of cutting planes appeared (*GOMORY [74]*)

several different types of cutting plane algorithms have been developed but none are clearly superior to all others. We developed an algorithm for solving the Master problem based on Gomory's original cuts *[26]*. The algorithm was based on a linear programming package devised by *MARSTEN [111]*. Unfortunately like many of the cutting plane algorithms on a variety of problems, it performed poorly because of the slow convergence of the algorithm. The explanation for this often (but by no means always) poor performance was given by *RUBIN [125]*.

This cutting plane algorithm was tested on several problems in order to solve the Master problem during Benders' decomposition procedure. When only a few Benders' cuts have been added, the cutting plane algorithm performed well since the optimal solution to the Master problem was found after only a few cutting planes were added to the problem. But as the Master problem grew bigger (because of the number of Benders' cuts that were added) the convergence started to be very slow. In fact the behaviour of the algorithm on problems with many Benders' cuts was very typical of that of cutting plane algorithms as reported by many researchers. After a few iterations (five iterations in the example) of the cutting plane algorithm the solution usually reached more than 95% of the optimal solution but after that the algorithm stopped improving to a point where it became almost stationary (as shown in Figure 3-12) although we know that from a theoretical point of view Gomory's algorithm should converge. Some authors suggested that at this point the algorithm should be stopped and then imbedded in a tree search where at each node of the tree the algorithm is used again. This was not attempted because of the development of a better way of deriving lower bounds to the master problem that can be included in a tree search as we will see later on.

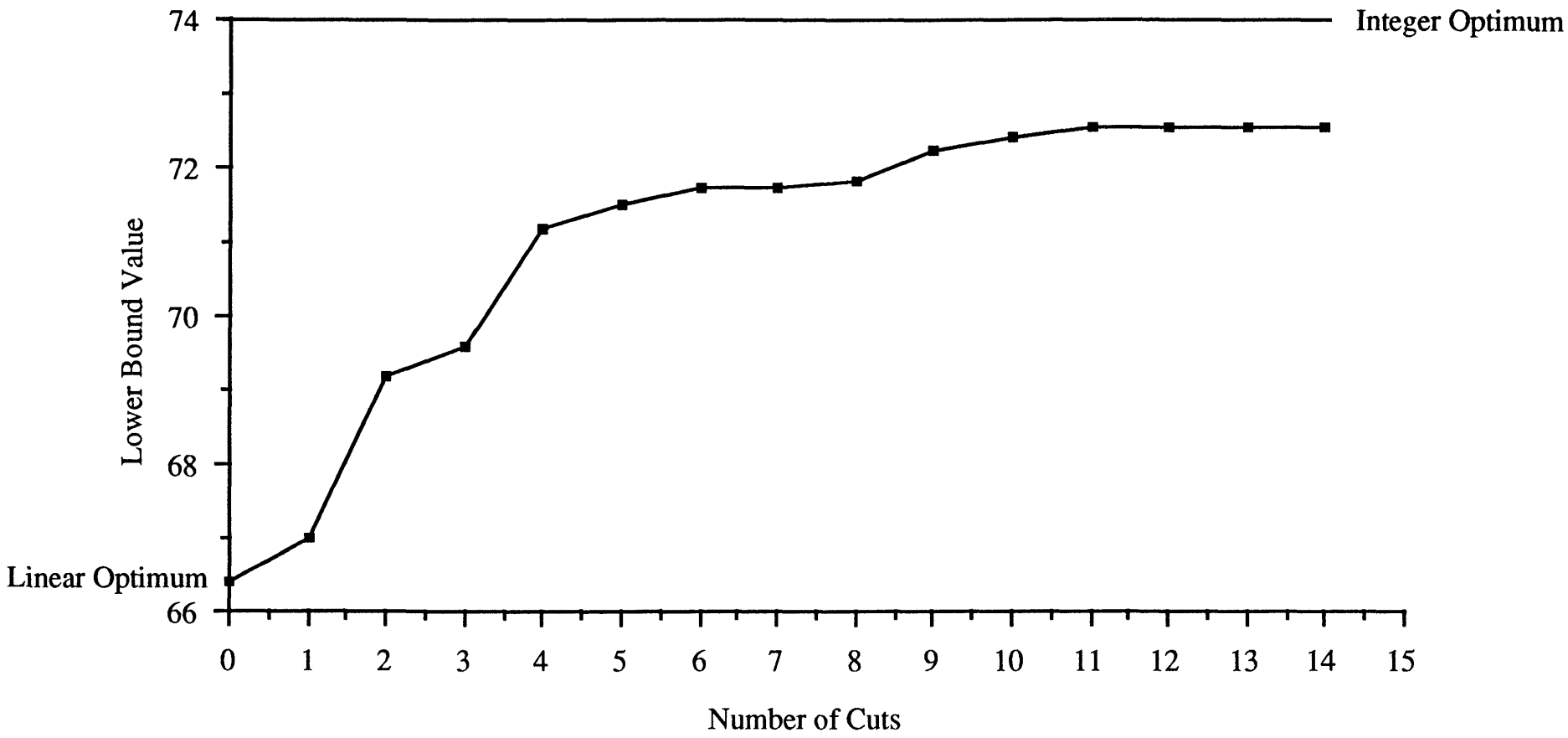


Figure 3-12 : Performance of the Cutting-Plane Algorithm
on problem test CST15x2

3-3-2 Tree search algorithm for the Master problem

The most successful technique that is widely used to solve such integer programming problems involves the use of enumeration techniques in such a way that not all feasible possibilities of the problem are enumerated. Only a few of them are generated the rest being implicitly considered and discarded because of certain rules. The above process has been given many names by different authors such as implicit enumeration, tree search, backtrack programming, or as it is best known branch and bound (*LITLE et al [105]*).

The term branch and bound , in fact coins exactly the process where one divides the problem in hand into subproblems , branches to one such subproblem and then by a bounding procedure finds a lower bound for that particular subproblem (in the case of a minimization problem). These lower bounds in combination with an upper bound on the whole integer programming problem being solved permit the elimination of further investigation of some of these subproblems until no such subproblems are left to be investigated and then the algorithm terminates.

This powerful technique has been the basis of most algorithms that have been developed to solve integer programming problems, and this case is no different.

3-3-3 Lower bounds to the Master problem

Before starting any description for a branch and bound algorithm, one needs to describe one important step of the whole technique. The computation of lower bounds associated with the subproblems created during the process of a branch and bound algorithm is of the most significant importance. We devote this section to the discussion of deriving lower bounds for the Master problem to be used in a branch and bound algorithm.

(A) An LP bound :

The first and easiest lower bound that one can derive is to relax the integrality constraints on problem (41)-(45). If we denote the relaxed problem by (LPZ) then we know that :

value of (LPZ) $\leq z^*$ where z^* is the optimal value for problem (44)-(48)

The XMP code of *MARSTEN [111]* was used to derive the value of (LPZ). This code was imbedded in a tree search where at each branching a new linear programming problem was solved to optimality in order to derive a lower bound for the subproblem being investigated.

Problem	LP structure rowsxcolumns	Bound at root node	Value of optimal solution	Gap (%) $(z_{Opt}-z_{LPZ})/z_{Opt}$	Execution time (secs)
CST9x2	19x27	64.74	65	0.48	0.121
CST15x2	23x37	66.40	74	10.27	0.171
CST20x2	27x41	76.62	86	10.91	0.241
CST25x2	34x51	72.05	78	7.63	0.343
CST10x3	21x40	89.76	91	1.36	0.546
CST15x3	27x55	106.41	127	16.21	0.641
CST20x3	29x61	126.13	140	9.91	0.752
CST25x3	32x76	172.24	201	14.31	0.787
CST15x4	20x61	179.65	221	18.71	1.102
CST20x4	26x81	193.82	242	19.91	1.31

Table 3-1 : LP bound for the Master problem at the Root node.
for a selection of 10 test problems (see Chapter 4).

The results on the sharpness of the bounds (as shown in Table 3-1) were very mixed

since in some cases the value of the LP relaxation was only 0.5% away from the the value of the optimum of the integer Master problem and in other cases it was as far as 20% away from the optimum. In the latter case very many nodes in the tree search were generated which had its effect on the computing time needed to achieve optimality. Bearing in mind that the Master problem is solved in this fashion several times (one full solution to the Master problem is needed for each Benders' cut added) it is clear that the lower bound is of great importance for the whole procedure.

(B) A bound based on Lagrangean Relaxation :

A natural extension to the above is to relax some other constraints of the problem instead of relaxing the integrality constraints. The relaxation is done in a Lagrangean fashion (*GEOFFRION [69]*) where one relaxes some constraints and leaves only those constraints that make the relaxed problem "easy" to solve. By "easy" we mean that we have tools that permit the solution of that relaxed problem in a relatively efficient way. Normally this means that the relaxed problem can be solved by a polynomial algorithm.

If we attach Lagrangean multipliers λ_i to constraints (45) and δ_c to constraints (47) we are left with the following problem :

$$\text{(LRZ)} \quad \min \left(1 + \sum_{c=1}^{P} \delta_c \right) z + \sum_{i=2}^{n} \sum_{k=1}^{M} \left[\lambda_i + \sum_{p=1}^{P} \gamma_{ik}^{p} \right] y_{ik} - V(\alpha, \beta, \lambda) \qquad (49)$$

subject to :

$$\sum_{i=2}^{n} q_i y_{ik} \leq Q_k \quad , \quad k = 1,...,M \qquad (51)$$

$$y_{ik} \in \{0, 1\} \quad , \quad i = 2,...,n \; ; \; k = 1,...,M \qquad (52)$$

where $V(\alpha,\beta,\lambda)$ is a constant given by :

$$\sum_{p=1}^{P}\left[\sum_{S}\alpha_S^p(|S|-1)+\beta^p(n-1)\right]+\sum_{i=2}^{n}\lambda_i$$

The above problem decomposes into M different 0-1 knapsack problems. Knapsack problems (although being in the NP-complete class) have been extensively studied and many "efficient" algorithms have been derived for them that run in polynomial time if certain numbers in the problem data are bounded (*MARTELLO and TOTH [112]*). These algorithms are often referred to as being pseudo-polynomial algorithms.

If z^* is an optimal value for problem (44)-(48) then it is well known that $v(LRZ) \leq z^*$ when a set of Lagrangean multipliers λ unrestricted and $\delta \geq 0$ are chosen.

It is clear that the best choice of these multipliers would be to choose λ and δ in such a way that it maximizes (49) i.e. :

$$\max_{\lambda,\delta}\left[\min\left[1+\sum_{p=1}^{P}\delta_c\right]+\sum_{i=2}^{n}\sum_{k=1}^{M}\left(\lambda_i+\sum_{p=1}^{P}\gamma_{ik}^p\right)y_{ik}\right]$$

Therefore, most schemes for determining λ and δ have as their prime objective the finding of an optimal solution to (49)-(51). One of the most successful schemes used in the last decade to get solutions to (49)-(51) is to use Subgradient Optimization (*FISHER [49] and SANDI [126]*). This is because of the ease of programming Subgradient Optimization and its good performance (*HELD et al [78]*).

Given an initial value for λ and δ say λ^0 and δ^0 then a sequence $\{\lambda^u\}$ and $\{\delta^u\}$ is generated by the following rule :

$$\lambda_i^{u+1}=\lambda_i^u+t_u\left[\sum_{k=1}^{M}y_{ik}^u-1\right]$$

$$\delta_p^{u+1}=\delta_p^u+t_u\left[z^u-\sum_{S}\alpha_S^c(|S|-1)-\beta^c(n-1)-\sum_{i}\sum_{k}\gamma_{ik}^c y_{ik}^u\right]$$

where y^u and z^u are values for the optimal solution to problem (LRZ_u).

t_u is a positive scalar step size computed as follows :

$$t_u = \frac{A_u\left(Z^* - Z_D(\lambda^u,\delta^u)\right)}{\sum_{i=2}^{n}\left(\sum_{k=1}^{M} y_{ik}^u\right)^2 + \sum_{c=1}^{P}\left(z - \left[\sum_{S}\alpha_S^c(|S|-1) + \beta^c(n-1)\right] - \sum_{i=2}^{n}\sum_{k=1}^{M}\gamma_{ik}^u y_{ik}^u\right)^2}$$

Z^* is an upper bound on (LRZ) frequently obtained by applying a heuristic to problem (44)-(48) and A_u is a scalar parameter which is initially set between 0 and 2 and is halved every few iterations if the bound has not improved (typically every 3 iterations which we applied to our case).

All the ingredients are now available to discuss the use of this bound in a branch and bound algorithm.

Table 3-2 shows the values of the Lagrangean Bound at the root node of the tree search for the previous 10 test problems. Table 3-3 compares the sharpness of the Linear Bound with the Lagrangean Bound. As we can see it from the latter the Lagrangean Bound is sharper than the Linear Bound in most of the cases. In fact this has been the feature of this algorithm on most of the test problems.

Figures 3-13 and 3-14 show the ascent of the Lagrangean Bound on two tests problems recorded at the root node where 50 iterations were performed.

Problem	Problem structure NxM	Bound at root node	Value of optimal solution	Gap (%) $(z_{Opt}-z_{LR})/z_{Opt}$	Execution time (secs)
CST9x2	10x2	63.86	65	1.75	0.27
CST15x2	15x2	72.23	74	2.39	0.88
CST20x2	20x2	84.73	86	1.48	1.67
CST25x2	25x2	75.91	78	2.68	2.13
CST9x2	10x3	88.86	91	2.35	0.32
CST15x3	15x3	123.66	127	2.63	1.27
CST20x3	20x3	134.71	140	3.78	2.33
CST25x3	25x3	213.41	201	2.61	3.14
CST15x4	15x4	213.41	221	3.43	1.71
CST20x4	20x4	231.38	242	4.39	3.28

Table3-2: LR Bound at the root node for a selection of 10 test problems

Problem	Optimum Solution	LP Bound	Gap (%)	LR Bound	Gap (%)	Difference LPZ - LR(%)
CST9x2	65	64.74	0.48	63.86	1.75	-1.27
CST15x2	74	66.40	10.27	72.23	2.39	+7.88
CST20x2	86	76.62	10.91	84.73	1.48	+4.95
CST25x2	78	72.05	7.63	75.91	2.68	+4.95
CST9x2	91	89.76	1.36	88.86	2.35	-1.14
CST15x3	127	106.41	16.21	123.66	2.63	+13.58
CST20x3	140	126.13	9.91	134.71	3.78	+6.13
CST25x3	201	172.24	14.31	195.76	2.61	+11.70
CST15x4	221	179.65	18.71	213.41	3.43	+15.28
CST20x4	242	193.82	19.91	231.38	4.39	+15.52

Table3-3: Comparison of the LR and LP bounds for the previous 10 test problems



Figure 3-13 : Ascent of the Lagrangean Bound on problem test CST15x2

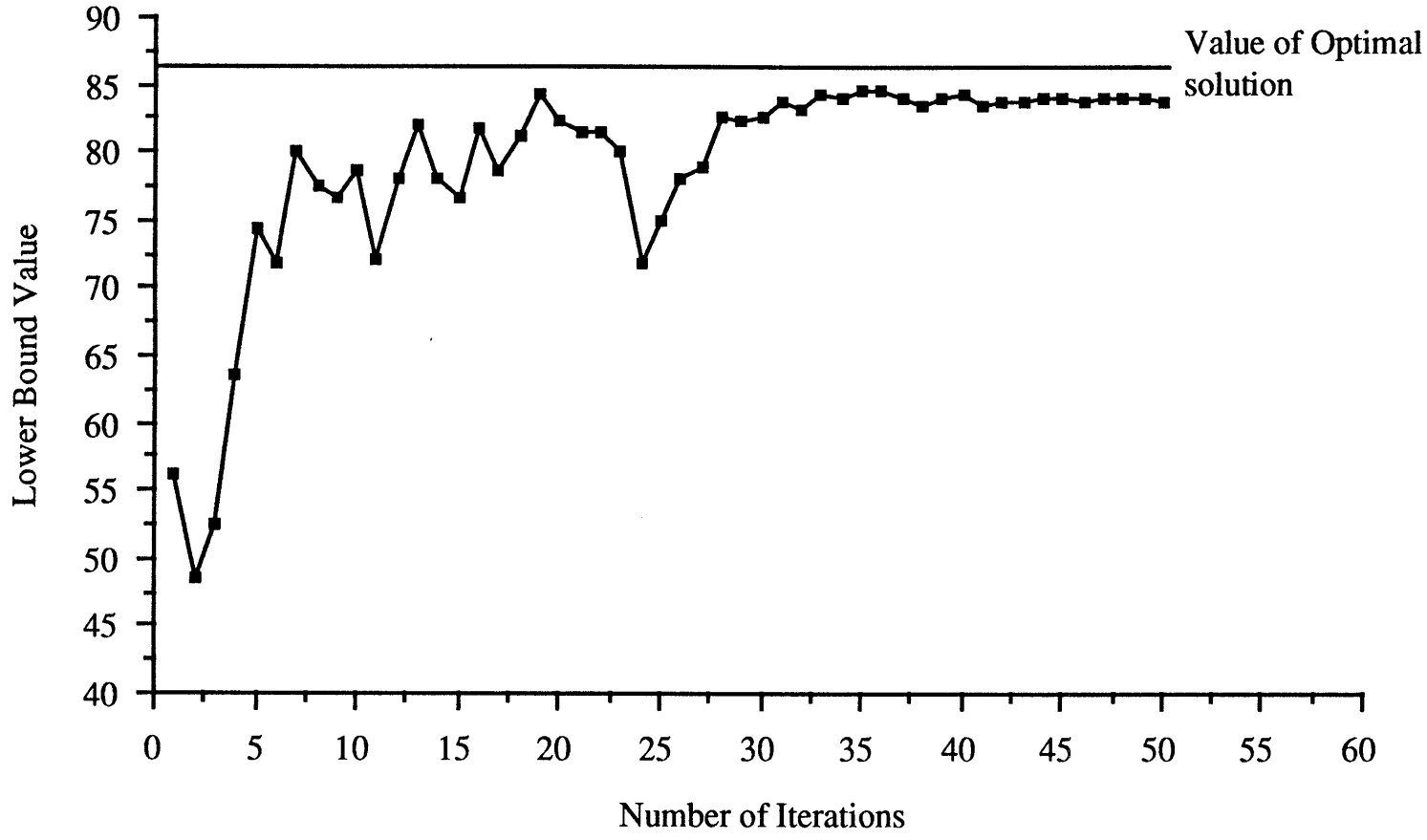


Figure 3-14 : Ascent of the Lagrangean Bound
on problem test CST20x2

3-3-4 A tree search based on a Lagrangean Relaxation bound

The Master problem (44)-(48) has been partitioned by the use of Lagrangean relaxation in such a way that the remaining constraints form a well structured problem that is "easy" to solve (i.e. like the knapsack problem). Lower bounds are then obtained by running several iterations of the Subgradient Optimization technique to get a bound that is as close to the optimum value of the Master problem as possible.

We have to remember that the Master problem is formulated in two sets of variables : one continuous variable z and binary variables y_{ik}. The Master problem decomposes itself after relaxation in two different problems that are not interlinked because constraints (47) are not present any more being relaxed in the objective function.

Therefore, we have to solve the following two separate problems :

$$\min \left(1 + \sum_{p=1}^{P} \delta_p \right) z \tag{52}$$

subject to : $z \geq 0$ (53)

and :

$$\min \sum_{i=2}^{n} \sum_{k=1}^{M} \left[\lambda_i + \sum_{p=1}^{P} \gamma_{ik}^{p} \right] y_{ik} \tag{54}$$

subject to :

$$\sum_{i=2}^{n} q_i y_{ik} \leq Q_k \quad , \quad k = 1,...,M \tag{55}$$

$$y_{ik} \in \{0,1\} \quad , \quad i = 2,...,n \; ; \; k = 1,...,M \tag{56}$$

The first problem is, in general, easy to solve : If $\left[1+\sum_{p=1}^{P}\delta_p\right] \geq 0$ then z = 0 is the solution otherwise z = + ∞

But in fact we know that z is the value of the optimal solution to the Master problem and that this value is always between two bounds in any tree search. The upper bound being the upper bound on the whole Master problem or whenever possible it is the best upper bound found during the tree search.

The lower bound is the lower bound on the whole problem i.e. the bound found by solving the previous Master problem. At any node of the tree search the bound found there constitutes a lower bound to the whole problem unless that node is fathomed.

The solution of the first problem at a node different from the root node is as follows : if $\left[1+\sum_{p=1}^{P}\delta_p\right] \geq 0$ then z = lower bound on the whole problem otherwise z = Upper bound on the Master problem being solved.

The second problem has to do with the solution of M separate 0-1 knapsack problems.

A tree search algorithm has been developed using this Lagrangean bound to solve the Master problem. The M different 0-1 knapsack problems are solved using the code developed by *MARTELLO and TOTH [113]* given in *SYSLO et al [132].*

A depth first search was used where the two branches from any node of the tree represent the assignment of a vertex i of the graph to a subtree k or not i.e y_{ik} = 1 or 0. With some y's thus fixed, the relaxed Master problem is solved in a Lagrangean fashion as described in the last section, with a number of 50 iterations at the root node and 5 iterations for any node generated in the tree search. Its solution yields a

lower bound on the value of the whole Master problem. Then comparisons with the best upper bound found to date are made in order to assess the viability of stopping from exploring further the tree from the current node by virtue of the fact that the lower bound at the present node can not exceed the value of the best upper bound found until now. If it does then that node is "fathomed" because the subproblems generated by branching from that node will never generate lower bounds that are lower than the present one. Backtracking is then necessary where one can still explore different nodes. The tree is terminated when such possible explorations are no longer possible.

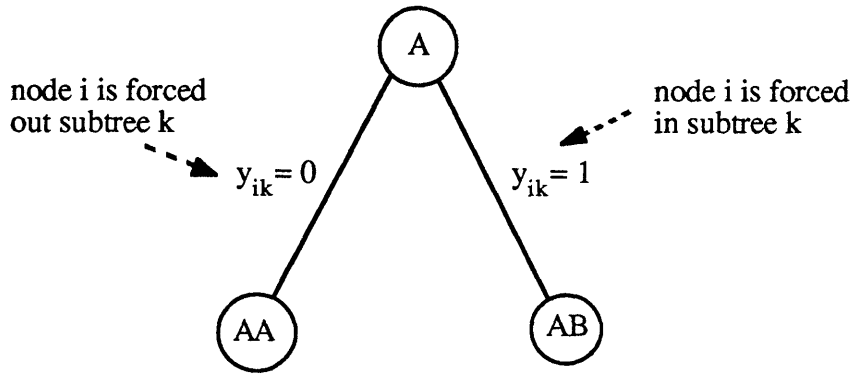


Figure 3-15 : Figure showing the branching tree.

Below is the outline of a conventional branch and bound algorithm applied to our special case :

z^*_{p-1} is the optimal value of the Master problem at iteration p-1 of the Benders' procedure.

STEP1 Initialization.

If p = 0 then $Z_b = 0$ otherwise $Z_b = z^*_{p-1}$

Get an upper bound for the Master problem by using a heuristic and call it Z_u.

STEP2 Root Node.

Determine a lower bound to the Master problem by solving the relaxed problem . This is achieved by solving problem (52)-(53) and problem (54)-(56).

Let Z_b be this lower bound.

- **If** $Z_b = Z_u$ STOP , the solution produced by the heuristic is optimal

- **Otherwise** partition node 0 into 2 nodes (see Figure 3-15).

- Set i = 1 and GOTO STEP3.

STEP 3 Examine node i.

Derive the corresponding problem from node i and then solve the relaxed problem by solving problem (52)-(53) and problem (54)-(56).

GOTO STEP 4.

STEP 4 Fathoming process.

- **If** $Z_b \geq Z_u$ then fathom node i

- **If** solution y_{ik} is feasible for problem (44)-(48) and Z_b is less than Z_u then $Z_u = Z_b$ and fathom node i.

- **If** node i is fathomed GOTO STEP 5 , **Otherwise** GOTO STEP 6

STEP 5 Backtracking Process.

- Backtrack to the parent node.

- Set i to be the first non-explored node and non-fathomed of the current nodes.

- **If** all nodes have been fathomed STOP. The optimal solution to the problem is the best feasible solution found so far **Otherwise** GOTO STEP 3.

STEP 6 Branching process.

- Branch following the rule :

The solution of the relaxed problem (52)-(53) amounts to solving M knapsack problems. This solution is most likely not feasible to constraints (45)-(46). Therefore, one picks one vertex that has been assigned at the same time to more than one subtree and use it as the next variable to branch on. This is done by choosing one such variable with maximum load.

If a variable y_{ik} is set to 1 then all other variables y_{jk} $j \neq i$ for that particular k are set to 0 otherwise the other branch is set to $y_{ik} = 0$ only (see Figure3-16).

- GOTO STEP 7

STEP 7 Partitioning process.

- Partition node i into two nodes i_1 and i_2 corresponding to a variable y_{ik} being set to 0 or 1.

- Set $i = i_1$ if i_1 is not fathomed, else $i = i_2$. (This corresponds to a depth first search.)

- GOTO STEP 3.

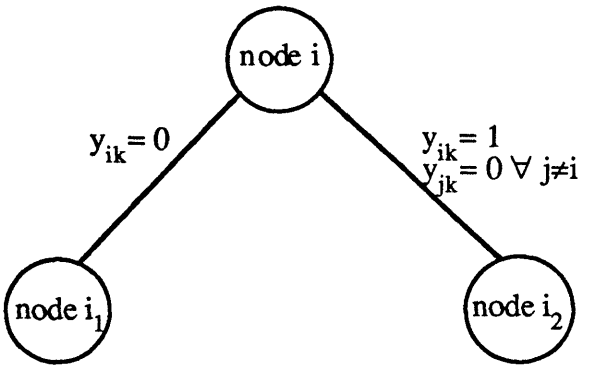


Figure 3-16 : Branching process

The tree search algorithm performed well in most cases where it yielded the optimal solution except for one case where after a certain time limit the optimal solution of the Master problem was not reached as it will be reported in the next Chapter which contains all the computational results. Below is Figure 3-17 which gives the nodes generated in the tree search of test problem 3 when the Master problem is solved in the second iteration of the Benders' procedure.

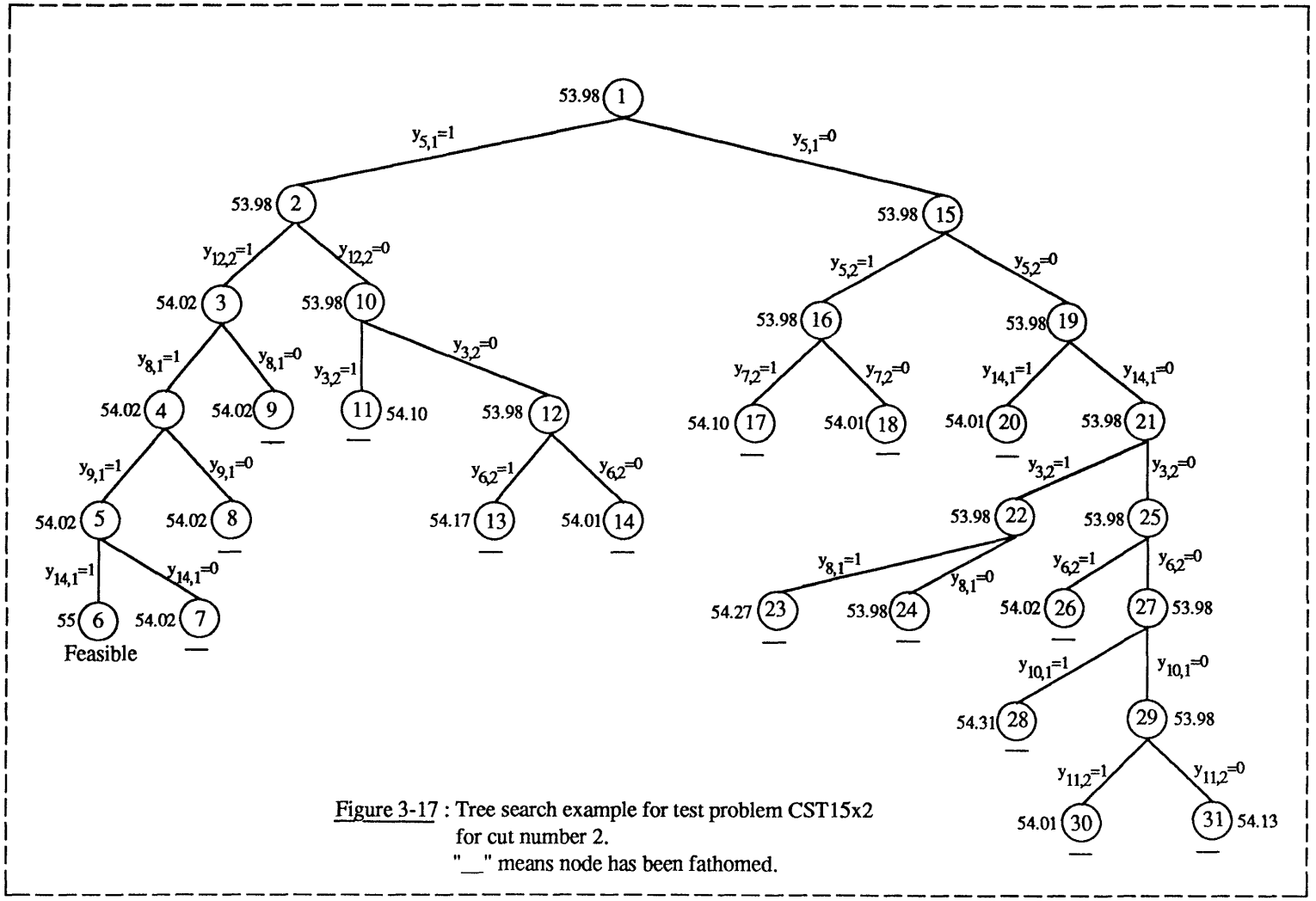


Figure 3-17 : Tree search example for test problem CST15x2
for cut number 2.
"__" means node has been fathomed.

3-4 UPPER BOUND FOR THE MASTER PROBLEM

The performance of a tree search depends on two main factors : the sharpness of the lower bound and that of the upper-bound.

The way we looked at the problem of finding an upperbound for the Master problem is that we need a feasible solution to (45)-(46) that satisfies constraints (47) with z minimum. In this sense it is quite normal to think about solving the P different problems in (57) and then taking z to be maximum of all the z^p. This gives a feasible solution to the Master problem that is an upper bound for it.

$$\left.\begin{aligned} z^p = \min \sum_{i=2}^{n} \sum_{k=1}^{M} \gamma_{ik}^p y_{ik} + V(\alpha^p,\beta^p) \\ \text{subject to:} \\ \sum_{k=1}^{M} y_{ik} = 1 \qquad i = 2,\ldots,n \\ \sum_{i=2}^{n} q_i y_{ik} \le Q_k \qquad k = 1,\ldots,M \end{aligned}\right\} p = 1,\ldots,P \qquad (57)$$

where $V(\alpha^p,\beta^p) = \sum_{S} \alpha_S^p(|S| - 1) + \beta^p(n - 1)$

It has to be said that this lends itself well to be solved in an iterative way after each Benders cut is added by just solving the corresponding problem z^p and taking the highest value between the previous upperbound to the Master problem and the value given by z^p. In fact solving (57) involves solving P different Generalized Assignment Problems (GAP's). The action taken instead was to use a heuristic to solve the GAP. The one used was the one derived by *MARTELLO and TOTH*

[113] to get an upperbound for the GAP problem. The results were satisfactory since the solution was never beyond 6% away from the optimum value of the Master problem.

3-5 CONCLUSIONS

This Chapter has been devoted to the material that concerns the solution of the Slave and Master problems. The first problem has been solved using a polynomially bounded algorithm. Some modifications were made in order to get a fully optimal solution to the Slave problem as applied to the programming formulation of the CST that was introduced in Chapter 2. The second problem was solved using the tools of Lagrangean Relaxation, branch and bound and Subgradient Optimization applied to the programming formulation of the Master problem as described in the beginning of this Chapter.

CHAPTER 4

COMPUTATIONAL RESULTS

4.1 INTRODUCTION

As we have been reporting in the last two Chapters, we introduced a new formulation for the Capacitated Spanning Tree problem (CST) which was well suited for solving with Benders' Decomposition Method. All the elements needed in order to solve that problem were reported in the last two Chapters and a computer algorithm was devised to test it.

In this Chapter we report on the computational results undertaken to test the algorithm devised in the two previous chapters. Also, we solve a numerical example by hand to show how the algorithm works. Data generation is also discussed.

4.2 NUMERICAL EXAMPLE

We have chosen a small example so that we can go through its solution by hand in order to understand the underlying structure for the algorithm.

The example is a 10 vertex problem and the relevant data is as follows :

- The cost matrix is given in Table 4-1.
- The customers requirements are given in Table 4-2.
- n = 10 is the number of vertices of the graph.
- M = 3 is the number of clusters we want to cluster the vertices in.

–	1	12	9	4	15	20	8	10	6
1	–	5	15	30	25	28	12	10	8
12	5	–	2	9	7	30	35	8	22
9	15	2	–	9	7	3	26	27	28
4	30	9	9	–	8	35	18	40	50
15	25	7	7	8	–	20	15	5	30
20	28	30	3	35	20	–	17	8	30
8	12	35	26	18	15	17	–	18	35
10	10	8	27	40	5	8	18	–	1
6	8	22	28	50	45	30	35	1	–

Table 4.1 : Cost matrix of the example.

	2	3	4	5	6	7	8	9	10
loads	5	7	12	11	8	10	9	2	14

Table 4.2 : Load requirements of the vertices.

We will be using this example to work out the algorithm developed in the last two Chapters in order to give to the reader a feel of how it works in practice.

For the sake of clarity and follow up we are going to present again the 0-1 formulation for the CST using the two sets of variables $x_{\ell k}$ and y_{ik} defined as follows :

$$x_{\ell k} = \begin{cases} 1 & \text{if arc } \ell \text{ belongs to the MST in subset } k \\ 0 & \text{otherwise} \end{cases}$$

$$y_{ik} = \begin{cases} 1 & \text{if vertex } i \text{ is in subset } k \\ 0 & \text{otherwise} \end{cases}$$

The problem can then be formulated as follows :

$$\min \sum_{\ell=1}^{m} \sum_{k=1}^{M} c_{\ell k} x_{\ell k} \tag{1}$$

subject to :

$$\sum_{i=2}^{n} q_i y_{ik} \leq Q_k \qquad k = 1,...,M \tag{2}$$

$$\sum_{k=1}^{M} y_{ik} = 1 \qquad i = 2,...,n \tag{3}$$

$$\sum_{\ell \in A(S)} \sum_{k=1}^{M} x_{\ell k} \leq |S| - 1 \qquad \forall S \subset N,\ 2 \leq |S| \leq n-1 \tag{4}$$

$$\sum_{\ell=1}^{m} \sum_{k=1}^{M} x_{\ell k} = n - 1 \tag{5}$$

$$\left.\begin{array}{l} x_{\ell k} \le y_{i_\ell k} \\ x_{\ell k} \le y_{j_\ell k} \end{array}\right\} \begin{array}{l} \ell = 1,\ldots,m \\ k = 1,\ldots,M \end{array} \qquad (6)$$

$$x_{\ell k},\ y_{ik} \in \{0,1\} \qquad \begin{array}{l} \ell = 1,\ldots,m \\ k = 1,\ldots,M \end{array} \qquad (7)$$

where i_ℓ , j_ℓ are the terminal vertices of arc ℓ, and A(S) is the set of arcs having both terminal vertices in S.

As discussed previously constraints (2)-(3) enforce the capacity constraint.

Constraints (4)-(5) are the mathematical formulation for the spanning tree.

And constraints (6) are the linking constraints.

For the numerical example given in Tables 4-1 and 4-2 the tailored Benders' algorithm for it will give the following steps and results :

STEP 0 Get a feasible solution to constraints (2)-(3).

This is done by using a heuristic to solve the Generalized Assignment Problem using constraints (2)-(3) and any objective function.

Let us take as a feasible solution to constraints (2)-(3) the following y^1_{ik} solution :

$y_{2,1} = 1$	$y_{8,2} = 1$	$y_{5,3} = 1$
$y_{3,1} = 1$	$y_{10,2} = 1$	$y_{6,3} = 1$
$y_{4,1} = 1$		$y_{7,3} = 1$
		$y_{9,3} = 1$

All the remaining y_{ij} are set to zero. The above means that vertices 2,3 and 4 belong to the same set namely the first set. Also that vertices 8 and 2 belong to the second set and the rest of the vertices i.e 5,6,7 and 9 belong to the third set.

Set LB = 0 (all the arcs cost are positive) and UB = + ∞

GOTO STEP 3.

1st iteration.

STEP 3 Solve the linear program generated with y fixed as given in STEP 0.

This y_{ik} when plugged in problem (1)-(7) generates a linear program which is defined on a new graph because the linking constraints (6) will set all the corresponding x_{ik} variables to 0 meaning that all these arcs can never belong to the optimal solution and therefore all the following arcs can be eliminated from the original graph :

(2,8), (2,10), (2,5), (2,6), (2,7), (2,9), (3,8), (3,10), (3,5), (3,6), (3,7), (3,9), (4,8), (4,10), (4,5), (4,6), (4,7), (4,9), (8,5), (8,6), (8,7), (8,9), (10,5), (10,6), (10,7), (10,8), (10,9).

–	1	12	9	4	15	20	8	10	6
1	–	5	15	–	–	–	–	–	–
12	5	–	2	–	–	–	–	–	–
9	15	2	–	–	–	–	–	–	–
4	–	–	–	–	8	35	–	40	–
15	–	–	–	8	–	20	–	5	–
20	–	–	–	35	20	–	–	8	–
8	–	–	–	–	–	–	–	–	35
10	–	–	–	40	5	8	–	–	–
6	–	–	–	–	–	–	35	–	–

Table 4-3 : Cost Matrix of original graph induced by y^1 fixed.

The solution of this linear program i.e of the MST on the above Table 4-3 is obtained by the algorithm described in Chapter 3 section 3-2 which gives

the following solution with the corresponding dual variables :

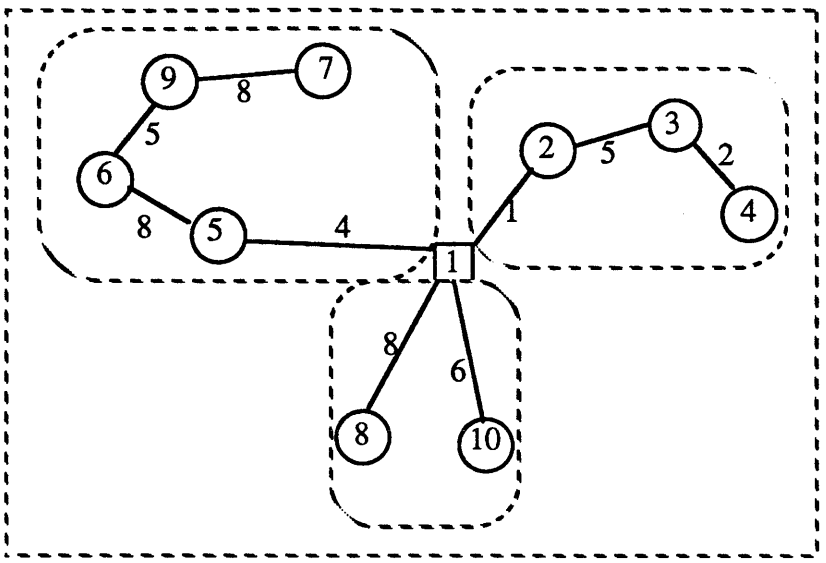


Figure 4-1 : MST solution of the Slave problem when $y = y^1$.

The value of this MST is 47 and therefore represents an upper bound on the value of the optimal value of problem (1)-(7). Consequently UB is updated to UB = 47 because $47 < \infty$

The dual variables are :

$\alpha_{S=\{1,2\}} = -3$

$\alpha_{S=\{3,4\}} = -3$

$\alpha_{S=\{1,2,5\}} = -1$

$\alpha_{S=\{1,2,3,4,5\}} = -1$

$\alpha_{S=\{6,9\}} = -3$

$\alpha_{S=\{1,2,3,4,5,10\}} = -2$

$\alpha_{S=\{1,2,3,4,5,6,9,10\}} = 0$

$\alpha_{S=\{1,2,3,4,5,6,7,9,10\}} = 0$

all the remaining α_S are set to 0.

Finally $\beta = 8$.

The γ_{tk} dual variables are then extracted from equations :

$$\sum_{S \supset \{i_\ell, j_\ell\}} \alpha_S + \beta + \gamma_{i_\ell k} \quad \leq \quad c_{\ell k}$$

as explained in Chapter 2 section 2-4 when α_S and β are known we take $y_{\ell k}$ to be :

$$\gamma_{\ell k} = \min\left[0 \ , \ (c_{\ell k} + \sum_{S \supset \{i_\ell, j_\ell\}} \alpha_S + \beta)\right]$$

In fact only the arcs that have been deleted will have any significance on these $\gamma_{\ell k}$ dual variables and therefore we get the following :

all $\gamma_{\ell k}$ are set equal to 0 except for the following arcs :

arc (3,6) where $\gamma_{\ell k} = -1$

arc (4,6) where $\gamma_{\ell k} = -1$

arc (4,7) where $\gamma_{\ell k} = -4$.

arc (9,10) where $\gamma_{\ell k} = -7$.

The above dual variables correspond to constraints (6) in the formulation which are of the form :

$x_{\ell k} \leq y_{i_\ell k}$ where i_ℓ , j_ℓ are the extremities of arc ℓ.

$x_{\ell k} \leq y_{j_\ell k}$ $\forall\, \ell = 1, \ldots, \dfrac{n(n-1)}{2}$ $\forall\, k = 1, \ldots, M$

From all these arcs and these $\gamma_{\ell k}$ dual variables we get the following Benders' cut :

$$z + y_{6,1} + 4y_{9,1} + 7y_{10,1} + y_{3,2} + 4y_{4,2} + y_{6,2} + 4y_{7,2} + 7y_{9,2} + y_{3,3} + 4y_{4,3} + 7y_{10,3} \geq 47$$

Add this Bender's cut to the Master problem and GOTO STEP2

STEP 2 Solve the Master problem.

The Master problem consists of the following problem :

min z

subject to :

constraints (2)-(3)

supplemented by any Benders's generated up to now.i.e. :

$$z + y_{6,1} + 4y_{9,1} + 7y_{10,1} + y_{3,2} + 4y_{4,2} + y_{6,2} + 4y_{7,2} + 7y_{9,2} + y_{3,3} + 4y_{4,3} + 7y_{10,3} \geq 47$$

$z \geq 0$, and $y_{ik} \in \{0,1\}$

We are at the first iteration and, therefore, the Master consists only of the capacity constraints (2)-(3) and the Benders' cut generated in STEP3. We solve this problem by the algorithm developed in Chapter 3 and get the following results :

$z = 23$

$y_{8,1} = 1$ $y_{3,2} = 1$ $y_{2,3} = 1$

$y_{9,1} = 1$ $y_{6,2} = 1$ $y_{4,3} = 1$

$y_{10,1} = 1$ $y_{7,2} = 1$ $y_{5,3} = 1$

all remaining y_{ij} equal to 0.

This y_{ij} solution becomes y^2 and $z = 23$ is a lower bound on the optimal value of problem (1)-(7).i.e we set LB = 23.

At this first iteration we have a lower bound on the problem LB = 23 and an upper bound UB = 47. Since they do not match we go to STEP 3 for another iteration.

2nd iteration

STEP 3 Solve Slave problem.

With y^2 as a fixed solution to constraints (2)-(3) we plug it in problem (1)-(7) we get a linear problem which is the slave problem that we have to solve.

Solution y^2 implies that the following arcs are to be eliminated from the

graph :

(8,3), (8,6), (8,7), (8,2), (8,4), (8,5), (9,3), (9,6), (9,7), (9,2), (9,4), (9,5), (10,3), (10,6), (10,7), (10,2), (10,4), (10,5), (3,2), (3,4), (3,5), (6,2), (6,4), (6,5), (7,2), (7,4), (7,5)

–	1	12	9	4	15	20	8	10	6
1	–	–	15	30	–	–	–	–	–
12	–	–	–	–	7	30	–	–	–
9	15	–	–	9	–	–	–	–	–
4	30	–	9	–	–	–	–	–	–
15	–	7	–	–	–	20	–	–	–
20	–	30	–	–	20	–	–	–	–
8	–	–	–	–	–	–	–	18	35
10	–	–	–	–	–	–	18	–	1
6	–	–	–	–	–	–	35	1	–

Table 4-4 : Cost matrix of the original graph induced by y^2.

The solution of this linear program on the above Table 4-4 by the algorithm described in Chapter 3 section 3-2 gives the following solution with the corresponding dual variables :

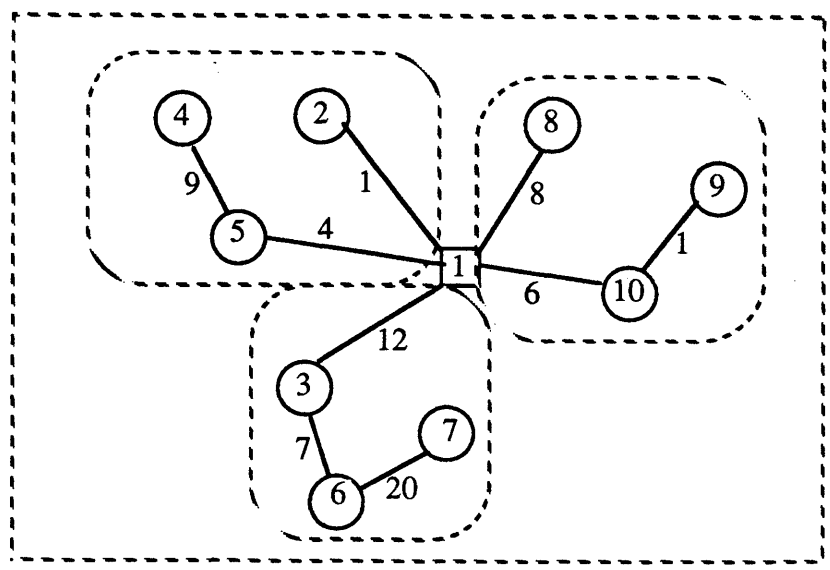


Figure 4-2 : MST solution of the Slave problem when $y = y^2$.

The value of the above MST is 68 and since UB = 47 < 68 we do not update UB. Since UB > LB we proceed with the algorithm.

The dual variables are :

$\alpha_{S=\{1,2\}} = -3$

$\alpha_{S=\{1,2,5\}} = -2$

$\alpha_{S=\{3,6\}} = -5$

$\alpha_{S=\{9,10\}} = -5$

$\alpha_{S=\{1,2,5,9,10\}} = -2$

$\alpha_{S=\{1,2,5,8,9,10\}} = -1$

$\alpha_{S=\{1,2,4,5,8,9,10\}} = -3$

$\alpha_{S=\{1,2,3,4,5,6,8,9,10\}} = -8$

and finally :

$\beta = 20.$

The γ_{tk} are then extracted as explained in the 1st iteration :

arc ((8,7) where $\gamma_{tk} = -3$

arc (9,3) where $\gamma_{tk} = -4$

arc (9,6) where $\gamma_{tk} = -7$

arc (9,7) where $\gamma_{tk} = -12$

arc (3,2) where $\gamma_{tk} = -7$

arc (3,4) where $\gamma_{tk} = -10$

arc (3,5) where $\gamma_{tk} = -3$

arc (6,4) where $\gamma_{tk} = -13$

arc (6,5) where $\gamma_{tk} = -4$

arc (7,4) where $\gamma_{tk} = -17$

With the above one generates a second Benders' cut :

$$z + 7y_{2,1} + 10y_{3,1} + 17y_{4,1} + 4y_{5,1} + 13y_{6,1} + 17y_{7,1} + 7y_{2,2} + 17y_{4,2} + 4y_{5,2} + 3y_{8,2} + 12y_{9,2} + 10y_{3,3} + 13y_{6,3} + 17y_{7,3} + 3y_{8,3} + 12y_{9,3} \geq 68$$

Add this Benders' cut to the Master problem and goto STEP 2

STEP 2 Solve the Master problem.

The Master problem now consists of the following problem :

min z

subject to :

constraints (2)-(3)

and Benders' cut 1 and cut 2

We solve this problem with the algorithm developed in Chapter 3 and get the following answer :

$z = 35$

$y_{6,1}= 1$ $y_{5,2}= 1$ $y_{2,3}= 1$

$y_{9,1}=1$ $y_{8,2}= 1$ $y_{3,3}= 1$

$y_{10,1}=1$ $y_{4,3}= 1$

$y_{7,3}= 1$

We update y^3 to be the above solution and proceed to STEP 3

3rd iteration :

STEP 3 Solve the Slave problem.

With y^3 fixed and plug in problem (1)-(7) we get the following cost matrix

–	1	12	9	4	15	20	8	10	6
1	–	5	15	–	–	28	–	–	–
12	5	–	2	–	–	30	–	–	–
9	15	2	–	–	–	3	–	–	–
4	–	–	–	–	–	–	18	–	–
15	–	–	–	–	–	–	–	5	30
20	28	30	3	–	–	–	–	–	–
8	–	–	–	18	–	–	–	–	–
10	–	–	–	–	5	–	–	–	1
6	–	–	–	–	45	–	–	1	–

Table 4-5 : Cost matrix of the original graph induced by y^3

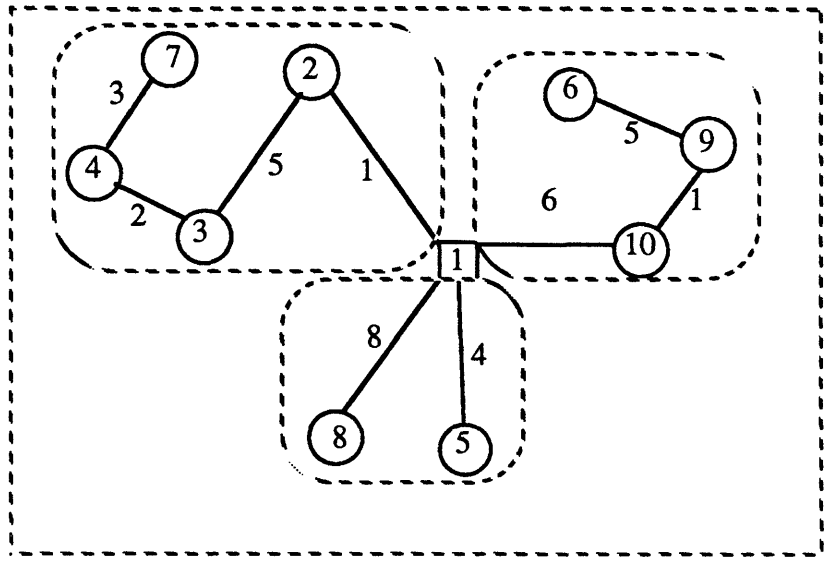


Figure 4-3 : MST solution of the Slave problem when $y = y^3$.

The cost of this MST is 35 and we then update UB to this value because the best UB that we have so far was 47 and, therefore, the algorithm terminates because UB = LB = 35. The optimal solution to the problem was found after only the addition of 2 Benders' cuts and is shown in Figure 4-3.

At this point there is no need in computing the dual variables because the algorithm has finished but if one wanted to look at the dual variables of the Slave problem that was solved in this step one would get the following :

$\alpha_{S=\{1,2\}} = -3$

$\alpha_{S=\{3,4\}} = -1$

$\alpha_{S=\{1,2,5\}} = -1$

$\alpha_{S=\{3,4,7\}} = -2$

$\alpha_{S=\{1,2,3,4,5,7\}} = -1$

$\alpha_{S=\{9,10\}} = -4$

$\alpha_{S=\{6,9,10\}} = -1$

$\alpha_{S=\{1,2,3,4,5,6,7,9,10\}} = -2$

and finally

$\beta = 8.$

We can see that the value of the objective function of the dual is always equal to the value of the primal problem (i.e Slave problem) since :

$9\times8 - (3\times1 + 1\times1 + 1\times2 + 2\times2 + 1\times5 + 4\times1 + 1\times2 + 2\times8) =$

$72 - 37 = 35.$

Figure 4-4 shows the convergence of the algorithm as the number of cuts are added to the Master problem.

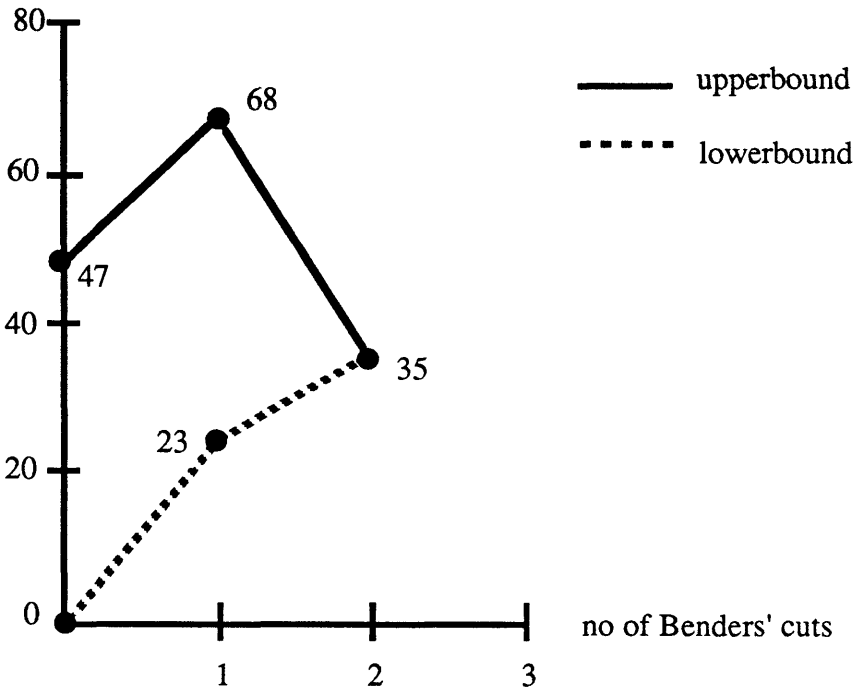


Figure 4-4 : Convergence of the Algorithm.

4.3 DATA GENERATION AND DESCRIPTION

The algorithm was tested on several tests problems that were generated randomly and some others that are Euclideans problems. The test problems have been classified into three categories. The first set of test problems is the set A of

randomly generated problems with loose capacity constraints. The second set of test problems is the set B of randomly generated problems with tight capacity constraints. The set B is obtained from set A by taking the same data except for the Q_k which are taken to be $0.8 \times Q_k$. This has the effect of tightening the capacity constraints which results in fewer feasible solutions for the Generalized Assignment Problem and some of the test problems which are feasible in data set A become unfeasible for data set B.

The last and third set of problems C has been a selection of 10 Euclidean problems taken from *Eilon and al [47]* whose description is given in Table 4-6.

Problem	Problem size		Capacity Constraint				Description
	n	M	Q_1	Q_2	Q_3	Q_4	
1	9	2	45	55	-	-	first 8 customers of the 100 customers problem on page 202 in Eilon et al [47]
2	11	2	65	75	-	-	first 10 customers of the 100 customers problem on page 202 in Eilon et al [47]
3	13	2	75	90	-	-	first 12 customers of the 100 customers problem on page 202 in Eilon et al [47]
4	15	2	100	120	-	-	first 14 customers of the 100 customers problem on page 202 in Eilon et al [47]
5	18	2	110	135	-	-	first 17 customers of the 100 customers problem on page 202 in Eilon et al [47]
6	20	2	120	145	-	-	first 19 customers of the 100 customers problem on page 202 in Eilon et al [47]
7	25	2	165	180	-	-	first 24 customers of the 100 customers problem on page 202 in Eilon et al [47]
8	9	3	30	35	40	-	first 8 customers of the 100 customers problem on page 202 in Eilon et al [47]
9	11	3	40	45	50	-	first 10 customers of the 100 customers problem on page 202 in Eilon et al [47]
10	13	3	50	60	70	-	first 12 customers of the 100 customers problem on page 202 in Eilon et al [47]
11	15	3	80	90	100	-	first 14 customers of the 100 customers problem on page 202 in Eilon et al [47]
12	18	3	80	90	100	-	first 17 customers of the 100 customers problem on page 202 in Eilon et al [47]
13	20	3	85	95	110	-	first 19 customers of the 100 customers problem on page 202 in Eilon et al [47]
14	25	3	110	120	130	-	first 24 customers of the 100 customers problem on page 202 in Eilon et al [47]
15	15	4	45	55	65	70	first 14 customers of the 100 customers problem on page 202 in Eilon et al [47]
16	20	4	70	80	100	120	first 19 customers of the 100 customers problem on page 202 in Eilon et al [47]

Table 4-6 : Description of test problems for data set C

For test problems in data set A and B the arc costs were randomly generated in the uniform interval [0,50] and the q_i were randomly generated in the interval [0,25]. The Q_k were taken as being equal to :

$$Q_k = \sum_{i=1}^{n} \frac{q_i}{M} + e$$

where e is a random number in the interval [-5,5]. Having in mind that the following inequality should always be true :

$$\sum_{i=1}^{n} q_i \leq \sum_{k=1}^{M} Q_k$$

4-4 COMPUTATIONAL RESULTS

The algorithm was written in Pascal and was tested on a CDC Cyber 855 machine. Of all the test problems that were solved exactly only few iterations (counting in tens of iterations as compared to hundred of iterations for some other authors *GAVISH [63]*) were needed in order to converge. However even if after a prespecified computing time the algorithm did not converge, there is always the possibility to have a feasible solution that is within a known percentage of the optimal solution. This is due to the fact that in this algorithm at any given iteration we always have a lower bound an upper bound on the optimal solution of the problem that we are solving.

The following tables give the results of the algorithm tests on several test problems of size varying between 9 and 25 vertices and for a number of subsets varying between 2 and 4.

DATA SET A

Problem Size		No. of problems	No. of Variables	No. of nodes in tree			Upperbound			Optimal Solution			Execution Time (secs.)		
n	M	solved	in Master problem	Ave.	Min.	Max.	Ave.	Min.	Max.	Ave.	Min.	Max.	Ave.	Min.	Max.
9	2	5	17	68	55	72	46	35	58	42	31	53	1.56	0.96	1.76
11	2	5	21	55	43	68	59	48	87	56	35	72	1.95	1.23	2.39
13	2	5	25	50	31	57	121	95	149	112	84	138	3.60	2.21	4.15
15	2	5	29	48	31	62	102	87	128	96	71	112	4.64	3.76	5.87
18	2	5	35	85	56	97	140	123	172	131	114	158	9.50	9.01	11.13
20	2	5	39	110	73	186	137	110	186	115	86	161	18.89	16.29	20.71
25	2	5	49	147	85	209	218	191	247	210	171	238	31.40	25.30	33.40
9	3	5	25	76	61	92	61	51	93	55	35	71	2.58	2.21	3.16
11	3	5	31	117	71	190	78	66	101	74	62	87	5.72	3.77	7.41
13	3	5	37	88	81	113	136	112	158	124	93	142	8.82	7.26	9.58
15	3	5	43	176	120	192	127	121	162	120	109	137	22.32	17.38	25.36
18	3	5	52	210	156	238	160	137	201	151	121	189	33.31	29.48	37.19
20	3	5	58	248	179	283	143	129	187	137	115	172	58.05	43.50	71.12
25	3	5	73	279	223	315	242	191	285	238	176	268	86.70	51.75	97.13
15	4	5	57	161	136	188	175	162	197	169	148	181	27.36	13.18	36.60
20	4	5	77	230	191	257	155	138	191	148	121	175	72.14	64.20	85.50
25	4	1	97	341	341	341	273	273	273	261	261	261	140.48	140.48	140.48

Table 4-8: Computational Results for the Master problem in the last iteration of the Benders' procedure.

DATA SET A

Problem Size		No. of problems	No. of arcs in Slave problem			Optimal Solution			Execution Time (secs.)		
n	M	solved	Ave.	Min.	Max.	Ave.	Min.	Max.	Ave.	Min.	Max.
9	2	5	27	24	29	58	51	67	0.14	0.10	0.16
11	2	5	43	41	46	67	53	78	0.28	0.23	0.31
13	2	5	62	58	65	121	92	147	0.33	0.30	0.35
15	2	5	96	92	99	121	84	139	0.51	0.49	0.53
18	2	5	131	125	136	152	131	167	0.74	.071	0.76
20	2	5	170	168	178	138	119	186	1.07	1.01	1.09
25	2	5	286	275	293	243	187	248	1.28	1.25	1.31
9	3	5	23	19	28	66	48	90	0.13	0.12	0.14
11	3	5	31	26	37	85	76	97	0.28	0.26	0.29
13	3	5	59	56	63	138	97	163	0.37	0.35	0.039
15	3	5	82	77	85	137	120	148	0.59	0.57	0.61
18	3	5	119	112	124	172	137	197	0.71	0.70	0.72
20	3	5	147	138	152	146	123	185	1.21	1.18	1.23
25	3	5	262	256	275	277	191	284	1.37	1.34	1.40
15	4	5	69	58	73	175	162	197	0.67	0.65	0.69
20	4	5	109	104	117	156	137	191	1.07	1.02	1.11
25	4	1	241	241	241	241	241	241	1.40	1.40	1.40

Table 4-9: Computational Results for the Slave problem in the last iteration of the Benders' procedure.

DATA SET A

Problem Size n	M	No. of problems solved	No. of Variables in each problem	No. of benders Cut Ave.	Min.	Max.	Total Execution Time (secs.) Ave.	Min.	Max.
9	2	5	88	4	2	6	5.62	2.34	8.47
11	2	5	120	9	7	12	17.49	8.67	43.22
13	2	5	154	5	3	8	18.33	4.87	51.32
15	2	5	238	7	5	12	23.41	10.36	73.46
18	2	5	340	9	7	16	66.95	46.23	94.51
20	2	5	418	12	8	19	87.16	74.86	121.33
25	2	5	648	10	9	18	110.41	91.12	163.40
9	3	5	132	5	2	7	32.16	17.46	44.10
11	3	5	195	6	5	8	35.66	23.41	55.96
13	3	5	231	6	5	7	51.42	25.51	81.40
15	3	5	357	9	8	14	96.39	62.41	150.63
18	3	5	510	14	12	14	188.49	146.67	220.64
20	3	5	627	15	11	19	221.56	177.33	340.51
25	3	5	972	16	12	18	432.15	331.40	552.42
15	4	5	476	11	9	18	136.23	110.33	197.10
20	4	5	836	17	15	22	522.54	446.33	667.73
25	4	1	1296	25	25	25	1211.13	1211.13	1211.13

Table 4-7: Computational results of the whole algorithm

DATA SET B

Problem Size		No. of problems	No. of Variables	No. of nodes in tree			Upperbound			Optimal Solution			Execution Time (secs.)		
n	M	solved	in Master problem	Ave.	Min.	Max.	Ave.	Min.	Max.	Ave.	Min.	Max.	Ave.	Min.	Max.
9	2	5	17	71	65	78	51	35	65	48	31	60	1.62	1.11	1.85
11	2	4	21	58	43	73	63	48	95	61	39	78	2.04	1.23	2.48
13	2	5	25	56	37	64	136	101	161	121	88	146	3.96	2.52	4.48
15	2	3	29	47	40	58	112	84	131	108	78	124	4.56	3.92	5.62
18	2	5	35	84	58	98	152	129	181	138	118	164	9.40	9.12	11.19
20	2	4	39	82	193	145	114	193	126	97	172	20.55	17.18	21.32	
25	2	5	49	171	88	221	231	201	252	227	192	243	36.20	26.37	34.17
9	3	5	25	81	65	94	67	43	86	61	39	80	2.73	2.37	3.30
11	3	2	31	128	84	211	89	74	112	78	73	92	6.211	3.94	8.25
13	3	5	37	118	93	137	131	111	167	138	102	1156	11.52	7.72	10.39
15	3	5	43	187	143	228	145	130	156	132	121	148	23.64	18.11	27.48
18	3	3	52	229	187	265	194	141	223	176	132	215	35.85	31.12	39.15
20	3	4	58	275	210	311	176	135	201	157	128	193	64.13	45.61	74.28
25	3	5	73	311	256	361	256	191	293	241	187	284	96.30	54.63	105.74
15	4	3	57	174	148	192	180	165	197	174	157	188	29.44	14.12	37.48
20	4	4	77	241	208	265	162	140	202	162	133	191	75.30	66.28	87.41
25	4	1	97	410	410	410	322	322	322	315	315	315	168.00	168.00	168.00

Table 4-11: Computational results of the Master problem in the last iteration of the Benders' procedure.

DATA SET B

Problem Size		No. of problems	No. of arcs in Slave problem			Optimal Solution			Execution Time (secs.)		
n	M	solved	Ave.	Min.	Max.	Ave.	Min.	Max.	Ave.	Min.	Max.
9	2	5	28	24	31	54	36	72	0.14	0.09	0.17
11	2	4	41	38	44	65	45	84	0.28	0.22	0.30
13	2	5	61	57	66	121	96	151	0.33	0.31	0.36
15	2	3	97	93	101	107	88	142	0.51	0.48	0.53
18	2	5	128	124	131	148	134	168	0.73	0.71	0.74
20	2	4	175	168	179	157	116	192	1.07	1.04	1.10
25	2	5	284	279	292	232	211	257	1.30	1.28	1.34
9	3	5	24	21	27	64	49	95	0.14	0.14	0.15
11	3	2	33	30	36	87	76	98	0.29	0.28	0.30
13	3	5	58	52	61	132	114	175	0.36	0.33	0.38
15	3	5	85	82	89	141	137	152	0.60	0.58	0.62
18	3	3	114	108	126	178	138	211	0.71	0.69	0.73
20	3	4	143	141	149	155	145	188	1.22	1.18	1.24
25	3	5	265	262	268	238	195	294	1.38	1.36	1.42
15	4	3	71	69	75	191	168	217	0.69	0.68	0.70
20	4	4	107	105	112	175	147	211	1.06	1.03	1.10
25	4	1	238	238	238	322	322	322	1.38	1.38	1.38

Table 4-12: Computational Results for the Slave problem in the last iteration of the Benders' procedure.

DATA SET B

Problem Size n	M	No. of problems solved	No. of Variables in each problem	No. of benders Cut Ave.	Min.	Max.	Total Execution Time (secs.) Ave.	Min.	Max.
9	2	5	88	7	4	10	11.34	8.56	14.34
11	2	4	120	8	7	12	16.32	12.52	19.36
13	2	5	154	8	6	11	31.68	24.51	36.78
15	2	3	238	14	10	18	63.84	39.41	75.71
18	2	5	340	12	9	16	112.80	87.30	126.42
20	2	4	418	15	12	19	308.25	156.15	368.33
25	2	5	648	13	10	21	470.60	287.13	523.75
9	3	5	132	7	6	8	45.11	28.13	52.94
11	3	2	195	10	7	13	62.10	57.10	67.10
13	3	5	231	8	5	11	92.16	75.23	98.56
15	3	5	357	12	7	17	283.68	210.30	322.40
18	3	3	510	17	13	21	357.85	256.57	395.75
20	3	4	627	16	12	19	641.13	341.20	775.12
25	3	5	972	18	15	23	963.34	730.42	1120.16
15	4	3	476	13	10	16	294.40	150.21	327.16
20	4	4	836	18	15	24	753.30	515.12	856.75
25	4	1	1296	34	34	34	3360.00	3360.00	3360.00

Table 4-10: Computational results for the whole algorithm

DATA SET C

Problem	Problem size n	M	No. of Variables in Master problem	No. of nodes in tree	Upperbound	Optimal Solution	Execution Time (secs.)
1	9	2	17	82	123	123	2.46
2	11	2	21	75	161	156	1.97
3	13	2	25	48	179	177	3.62
4	15	2	29	61	196	192	5.82
5	18	2	35	151	235	224	16.13
6	20	2	39	132	237	231	21.36
7	25	2	49	268	289	285	55.74
8	9	3	25	115	141	137	3.86
9	11	3	31	223	168	165	10.58
10	13	3	37	158	198	193	15.22
11	15	3	43	215	207	201	27.48
12	18	3	52	256	248	241	39.90
13	20	3	58	312	266	262	80.45
14	25	3	73	375	315	311	115.50
15	15	4	57	223	237	232	37.58
16	20	4	77	415	324	317	127.51

Table 4-14: Computational results for the Master problem in the last iteration of the Benders' procedure.

DATA SET C

Problem	Problem Size n	M	No. of arcs in Slave problem	Optimal Solution	Execution Time (secs.)
1	9	2	29	138	0.12
2	11	2	41	161	0.27
3	13	2	66	177	0.31
4	15	2	92	212	0.48
5	18	2	136	234	0.71
6	20	2	172	236	0.95
7	25	2	287	296	1.22
8	9	3	24	145	0.11
9	11	3	33	188	0.26
10	13	3	58	223	0.35
11	15	3	84	242	0.51
12	18	3	121	266	0.69
13	20	3	152	311	1.12
14	25	3	261	365	1.33
15	15	4	72	247	0.61
16	20	4	113	348	1.01

Table 4-15: Computational results for the Slave problem in the last iteration of the Benders' procedure.

DATA SET C

Problem #	Problem size n	M	No. of Variables in the problem	No. of benders Cut	Total Execution Time (secs.)
1	9	2	88	7	12.31
2	11	2	120	5	7.89
3	13	2	154	7	18.10
4	15	2	238	13	58.34
5	18	2	340	10	112.93
5	20	2	418	14	234.85
6	25	2	648	13	501.67
7	9	3	132	12	30.98
8	11	3	195	13	95.22
9	13	3	231	10	106.58
10	15	3	357	9	137.49
11	18	3	510	18	552.78
12	20	3	627	14	804.53
13	25	3	972	24	2079.66
14	15	4	476	21	601.28
15	20	4	836	28	2932.71

Table 4-13: Computational results for the whole algorithm

4-5 COMMENTS

As mentioned earlier the Benders' method applied to this specific case enjoyed a very fast rate of convergence in most of the test problems. The only drawback of this method is the amount of time spent trying to find an optimal solution to the Master problem. Nonetheless one can be satisfied that the algorithm performed well in most of the cases.

CHAPTER 5

A FORMULATION FOR THE CST BASED ON SET PARTITIONING

5-1 INTRODUCTION

As we have seen in previous Chapters the Constrained Spanning Tree problem (CST) was formulated as a 0-1 integer programming problem that was well suited to use of Benders' Decomposition.

In this Chapter we report on a new 0-1 integer programming formulation of the CST based on different concepts than those reported in earlier Chapters along with some of the computational results achieved.

5-2 A 0-1 INTEGER PROGRAMMING FORMULATION FOR THE CST

We consider a graph G=(V,A) where V the set of vertices $v_0,v_1,...,v_n$; v_0 representing the root (central facility), n+1 being the total number of vertices of the whole graph and A the set of arcs. Each vertex $v \in V$ has a load q_i associated with it except for the root v_0.

If we make the (reasonable) assumption that the Capacitated Spanning Tree problem (CST) has the same capacity load Q for each of the M subsets that we want to cluster the vertices in, then we can make use of a very interesting formulation derived for the vehicle routing problem by *BALINSKI and QUANDT [7].*

That formulation was successfully used by *CHRISTOFIDES et al [31]* and subsequently by *LUCENA [105]* to solve the vehicle routing problem (VRP). The following is the adaptation of that formulation to our specific case.

Let us assume that we have a set $T_0=\{t_1,t_2,...,t_r\}$ of all possible trees (subtrees) rooted at vertex v_0. Let the total load on tree $t \in T_0$ be equal to $Q_t \leq Q$, and let d_t be its cost.

Let $N_i \subset T_0$, where N_i is the subset of all subtrees rooted at v_0 containing vertex i.

Let :

$$y_t = \begin{cases} 1 & \text{if subtree t appears in the optimal solution} \\ 0 & \text{otherwise} \end{cases}$$

A formulation of the CST is then given by :

$$\min \sum_{t \in T_0} d_t y_t \qquad (1)$$

subject to :

$$\sum_{t \in N_i} y_t = 1 \qquad \forall i \in V - \{v_0\} \qquad (2)$$

$$y_t \in \{0,1\} \qquad \forall t \in T_0 \qquad (3)$$

Constraints (2) ensure that every vertex belongs to exactly one subtree.

Constraints (3) are the integrality constraints.

Formulation (1)-(3) is that of a Set Partitioning Problem (SSP) (*BALAS and PADBERG [4]*) so any algorithm for solving the SSP can then be applied to solve (1)-(3) provided the number of variables involved is reasonably small (not more than 4-5 thousand variables).

In fact even for moderately sized problems, one should expect that the number of subtrees rooted at vertex v_0 and of total load requirement less than or equal to Q, be enormous. Also we note that no mention has been reported on the number of clusters one might want in the graph when solving the CST.

Following the same ideas of *CHRISTOFIDES et al [31]* we introduced another constraint to formulation (1)-(3) in order to take into account that we need in fact M clusters. The additional constraint is :

$$\sum_{t \in T_0} y_t = M \qquad (4)$$

Formulation (1)-(4) above is referred to as the Cardinality Constrained Set Partitioning Problem (CCSPP).

As discussed earlier, the number of variables involved in this kind of formulation is usually huge. The first thing one can do is to try to reduce the number of variables that one has to deal with.

In the next sections we are going to discuss the use of formulation (1)-(4) to solve the CST optimally. This is done by means of relaxing formulation (1)-(4) so that the number of variables is reduced considerably which will allow the use of an equivalent formulation to (1)-(4) in order to derive lower bounds for the CST. Also some reduction tests are developed to further eliminate more variables which will enable us at the end to develop a tree search algorithm for formulation (1)-(4) that can solve practical sized problems.

5-3 RELAXATIONS AND LOWER BOUNDS FOR THE CST

Although formulation (1)-(4) is very compact, it has too many variables. We are going to discuss relaxations and reduction tests that will enable us to transform this formulation in such a way that we would be able to use explicitly the new formulation to derive lower bounds for the CST.

5-3-1 Lower bounds for the CST

Let us consider the CST formulation in (1)-(4) and assume that the subtrees in T_0 are ordered in descending value of their loads. A load of a subtree is defined as being the sum of the loads of the vertices that form that subtree.

Let us introduce the following ξ_{it} variables defined as follows :

$$\xi_{it} = \begin{cases} 1 & \text{if vertex i belongs to subtree t in the optimal solution} \\ 0 & \text{otherwise} \end{cases}$$

Then we can link variable y_t of formulation (1)-(4) to variable ξ_{it} to get the following

equation :

$$y_t = \frac{1}{Q_t} \sum_{i \in M_t} \xi_{it} q_i \qquad (5)$$

where M_t is the set of all vertices of subtree t.

We then replace $y_t \; \forall \; t \in T_0$ in (1)-(4) by the RHS of (5) and introduce the following constraint :

$$\xi_{it} = \frac{1}{Q_t} \sum_{j \in M_t} \xi_{jt} q_j \qquad \forall \; i \in M_t, \; \forall \; t \in T_0 \qquad (6)$$

The above constraint (6) arises from the fact that if a given vertex is going to belong to a certain subtree t in the optimal solution then all the vertices that belong to that same subtree should be in the same subtree in the optimal solution.

with the introduction of variables ξ_{it} formulation (1)-(4) becomes as follows :

$$\min \sum_{t \in T_0} \frac{d_t}{Q_t} \sum_{i \in M_t} \xi_{it} q_i \qquad (7)$$

subject to :

$$\sum_{t \in N_i} \xi_{it} = 1 \qquad \forall \; i \in V - \{v_0\} \qquad (8)$$

$$\xi_{it} = \frac{1}{Q_t} \sum_{j \in M_t} \xi_{jt} q_j \qquad \forall \; i \in M_t, \; \forall \; t \in T_0 \qquad (9)$$

$$\sum_{t \in T_0} \sum_{i \in M_t} \frac{q_i}{Q_t} \xi_{it} = M \qquad (10)$$

$$\xi_{it} \in \{0,1\} \qquad \forall i \in V - \{v_0\}, \forall t \in T_0 \qquad (11)$$

Formulation (7)-(11) is equivalent to formulation (1)-(4) but still the number of variables involved is huge. However, this formulation allows the possibility to reduce the number of variables by relaxing some constraints. In parallel with the work of *CHRISTOFIDES et al [31]* when working with the VRP, we followed the same procedure by relaxing constraints (9) and replacing the set M_t by the entire set $V-\{v_0\}$. In this way the resulting formulation had a much reduced number of variables compared to the original one (1)-(4). Also, this meant that the value of the optimal solution of (7)-(11) with (9) relaxed and M_t replaced by $V-\{v_0\}$ will yield a lower bound on the value of the optimal solution of formulation (1)-(4).

It is clear that for any subtree t to be considered in the final solution it needs to have $Q_t \leq Q$. On the other hand since we need M clusters then we can say that there is also a lower bound which a subtree should achieve in terms of load in order to be considered in the final solution. This lower bound is as follows :

$$q^* = \sum_{i \in I} q_i - Q(M-1)$$

Therefore, the set of feasible loads that any subtree t should attain is given by :

$$F = \{q^*, q^*+1, q^*+2, \ldots, Q\}.$$

The relaxation of constraint (9) and the replacement of M_t by $V-\{v_0\}$ meant that for each vertex i and for each feasible load $\ell \in F$ we only need to consider the minimum cost subtree rooted at v_0 of load ℓ containing that vertex i.

With the above definitions and results let us introduce a variable $\xi_{i\ell}$ defined as follows :

$$\xi_{i\ell} = \begin{cases} 1 & \text{if vertex i is in a subtree of load } \ell \\ 0 & \text{otherwise} \end{cases}$$

The introduction of the variable $\xi_{i\ell}$ instead of ξ_{it} reduces the number of variables from considering the combination of each vertex i for each subtree t, to considering the combination of each vertex i for each feasible load ℓ. As the number of possible subtrees t of total load $Q_t \leq Q$ is far greater than that of feasible loads ℓ, the number of variables is reduced considerably.

Then we can rewrite (7)-(11) after dropping (9) and replacing M_t by $V - \{v_0\}$:

$$\min \sum_{i \in I} \sum_{\ell \in F} \tilde{d}_{i\ell} \xi_{i\ell} \qquad (12)$$

subject to :

$$\sum_{\ell \in F} \xi_{i\ell} = 1 \qquad \forall i \in V - \{v_0\} \qquad (13)$$

$$\sum_{i \in I} \sum_{\ell \in F} \left(\frac{q_i}{\ell} \right) \xi_{i\ell} = M \qquad (14)$$

$$x_{i\ell} \in \{0,1\} \qquad \forall i \in V - \{v_0\}, \forall \ell \in F \qquad (15)$$

where $\tilde{d}_{i\ell} = d_{i\ell}\left(\frac{q_i}{\ell} \right)$.

and $d_{i\ell}$ is the cost of the minimum cost spanning tree that contains vertex i and is of load ℓ.

Therefore, the above formulation (12)-(15) is a relaxation on the original formulation (1)-(4) when the values of $d_{i\ell}$ are known and formulation (12)-(15) is solved to optimality. In other words :

Value of optimal solution of (1)-(4) $\geq$ value of optimal solution of (12)-(15). Still one is left with the problem of generating for each vertex $i \in V - \{v_0\}$ and for each feasible load $\ell \in F$ a tree rooted at v_0 and of minimum cost.

5-3-2 Getting lower bounds on the values of the $d_{i\ell}$

The problem of finding the minimum cost subtree t rooted at v_0 and of load ℓ is not a trivial one. This is because of the exponential number of feasible subtrees containing vertex $i \in V - \{v_0\}$ and rooted at v_0 for every feasible load $\ell \in F$.

We can recall that we are given a graph G consisting of a set of arcs A and a set of vertices V (up to now all the graphs considered have been complete ones), with load requirement q_i on every vertex $i \in V$ except the root vertex v_0 which has load 0.

Therefore, for a given vertex i and a feasible load ℓ we want to find in the graph G a minimum cost subtree t containing vertex i and of total load requirement ℓ.

This is in fact no more than the well known Steiner Problem plus one additional constraint. The general Steiner Problem can be stated as follows :

Given a graph G=(V,A) and a subset $S \subset V$, the problem is then to find a subtree t in G that spans all the vertices contained in S at minimum cost. Clearly the subtree might encompass more vertices than those of S only.

The similarity between the Steiner Problem and our previous problem is that we are looking for a subtree that spans vertices i and v_0 at a minimum cost plus the additional constraint that the total load requirement equals a fixed value ℓ known in advance. If the latter constraint was not to be enforced then the problem reduces to finding for the shortest path between v_0 and i.

The Steiner Problem is known to be an NP-hard problem to solve (*GAREY et al [60]*).

BEASLEY [13] gave a very interesting formulation for the Steiner Problem based

on a Shortest Spanning Tree formulation.

Let us first give some details of the proposed formulation for the Steiner Problem to which we will later on add an additional constraint to make it correspond to the problem of solving the minimum cost subtree rooted at vertex v_0 containing vertex i and of total load requirement ℓ.

Let V and A be defined as above on graph G.

Let K be the set of vertices which are to be connected together ($K \subset V$) (In our case K consists of vertices v_0 and i only).

Let c_{ij} be the cost of arc (i,j) ($c_{ij} \geq 0$).

The Steiner problem is to chose a subset of A which connects together all the vertices of K at a minimum cost. It is well known that arcs in this subset of A will form a minimum cost spanning tree (called the Steiner tree) on some set of vertices $K \cup S$ where $S \subset V - K$ is called the set of Steiner vertices.

To motivate his formulation *BEASLEY [13]* introduced the following transformations to derive a graph $\tilde{G}$ from G :

(a) add an artificial vertex (say vertex $v_{0'}$) to graph G.

(b) for each vertex $i \in V\text{-}K$ add an arc $(v_{0'},i)$ of cost 0.

(c) for vertex $v_0 \in K$ add an arc $(v_{0'},v_0)$ of cost 0

(d) find the MST of the resulting graph $\tilde{G}$ subject to the additional restriction that in that MST any vertex $i \in V - K$ connected by the arc $(v_{0'},i)$ to vertex $v_{0'}$ have vertex degree 1.

Then it is clear that the optimal solution to this restricted MST problem will be of the following form :

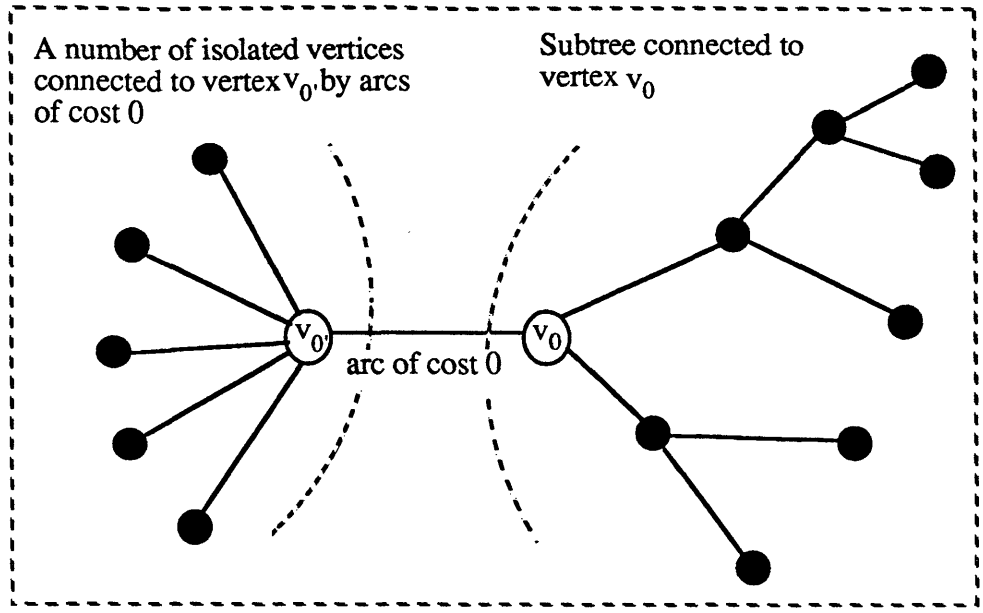


Figure 5-1 : Tree structure after solution of constrained MST on graph $\tilde{G}$

The claim made and proved in *[13]* is that the subtree connected to vertex v_0 in Figure 5-1 is the Steiner tree (i.e. the optimal solution to the Steiner Problem on the original graph).

From this a mathematical programming formulation of the restricted MST on the graph $\tilde{G}$, derived by the transformations (a)-(c) cited above, is equivalent to a formulation of the Steiner Problem on the original graph G.

Let V' represent $V \cup \{v_{0'}\}$

Let $A_{0'}$ represent $A \cup \{(v_{0'},i) / i \in V\text{-}K \cup \{v_0\}\}$ and $c_{0'i} = 0 \; \forall \; i \in V\text{-}K \cup \{v_0\}$.

Define $P_i = \{(i,j) / (i,j) \in A\} \cup \{(j,i) / (j,i) \in A\}$. P_i represents the arcs of A which involve vertex i.

Let x_{ij} be defined as follows :

$$x_{ij} = \begin{cases} 1 & \text{if arc } (i,j) \text{ is in the optimal solution} \\ 0 & \text{otherwise} \end{cases}$$

Then the formulation of the Steiner Problem is given by :

$$\min \sum_{(i,j)\in E_{0'}} c_{ij} x_{ij} \qquad (16)$$

subject to :

$$\text{the } x_{ij} \text{ form a spanning tree on } \tilde{G} \qquad (17)$$

$$x_{0'i} + x_{pq} \leq 1 \qquad \forall (p,q) \in P_i \qquad (18)$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in V' \qquad (19)$$

Equation (18) ensures that if arc $(v_{0'},i)$ is used, no other arc which involves i can be used.

This was a clever idea to reduce the Steiner Problem to a much easier problem when constraints (18) are relaxed in a Lagrangean way. This problem is in fact the well known Minimum Spanning Tree problem (MST).

However, in our case we want a subtree t rooted at vertex v_0 containing vertex i (i.e $K=\{v_0,i\}$) plus the additional constraint that the load requirement of t be equal to a given $\mathcal{L}$.

We can accommodate for this by introducing a new variable $y_{0'i}$ such that :

$$y_{0'i} = \begin{cases} 0 & \text{if arc } (v_{0'},i) \text{ is chosen in the MST} \\ 1 & \text{otherwise} \end{cases}$$

Therefore, the capacity constraint can be expressed as follows :

$$\sum_i q_i y_{0'i} = \mathcal{L}$$

Looking at the definition of the x_{ij} we can see that :

$$y_{0'i} = 1 - x_{0'i}$$

Using the above mathematical programming formulation of the Steiner problem and the additional capacity constraint the problem of finding a minimum cost subtree t rooted at v_0 containing vertex i and of total load capacity ℓ can be mathematically formulated in one set of variables only namely x_{ij} :

$$\min \sum_{(i,j)} c_{ij} x_{ij} \tag{20}$$

subject to :

$$(x_{ij}) \text{ form a spanning tree} \tag{21}$$

$$x_{0'i} + x_{pq} \leq 1 \qquad \forall (p,q) \in P_i \quad \forall i \in V - K \tag{22}$$

$$\sum_i (1 - x_{0'i}) q_i = \ell \tag{23}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in V' \tag{24}$$

This problem is as hard to solve as the Steiner Problem but this formulation lends itself to be used in a way such as to obtain lower bounds on the $d_{i\ell}$ by relaxing constraints (22) and (23) in a Lagrangean fashion and then solving the remaining problem which an MST. We draw the attention on the fact that the above procedure has to be applied for every vertex i and every load $\ell \in F$ to get all the lower bounds on the $d_{i\ell}$.

In the process of solving the CST we characterized a new problem which is that of finding a minimum cost spanning subtree rooted at vertex v_0 containing vertex i and

of total load requirement ℓ. An integer mathematical programming formulation was derived for it that if solved to optimality will get the exact values of the $d_{i\ell}$.

5-3-3 Generation of all feasible subtrees rooted at v_0

In fact we did not use formulation (20)-(24) to get lower bounds on the $d_{i\ell}$ but instead derived the exact value of the $d_{i\ell}$ by generating all the subtrees of load less or equal to Q for each feasible load ℓ. Then for each vertex i and for each feasible load ℓ we extracted the corresponding minimum cost spanning subtree rooted at v_0. This is because it was more attractive from a computing point of view to generate all possible subtrees $t \in T_0$ than to compute MST several times just to get lower bounds on the $d_{i\ell}$. It is. of course easy to think of problems where the above statement would not be true.

Every vertex of graph G has a load requirement q_i. If we consider these requirements to be n given numbers and that we are given a number ℓ then we are looking for all possible combinations of the n numbers that sum to ℓ. This problem is at the root of several interesting problems in number theory, operations research and polynomial factorization (*HOROWITZ and SAHNI [80]*). Relating to our problem this will give us all feasible subsets of the set $V - \{v_0\}$ that sum up to ℓ for any $\ell \in F$. This number is going to be a huge number because of the exponential nature of the problem. But subexponential algorithms have been derived together with some heuristics to make the generation of those subsets relatively easy in terms of computing terms. During the generation process one can compute all the MST corresponding to all the sets and then generate a table that gives for each vertex i and each feasible ℓ the minimum cost spanning subtree rooted at v_0 containing vertex i and of total load requirement equal to ℓ. This will enable us to compute lower bounds for the CST problem.

Let us first look at an algorithm that will allow us to generate all feasible partitions for the q_i that will sum up to a given ℓ. In *HOROWITZ and SAHNI [80,81]* one such algorithm is derived to do exactly what we are looking for. Below is the outline of this algorithm.

Let us suppose that the requirements q_i are ordered in increasing order i.e. $q_1 \leq q_2 \leq \cdots \leq q_n$.

We define x_i as follows :

$$x_i = \begin{cases} 1 & \text{if element } q_i \text{ is chosen to be in the subset} \\ 0 & \text{otherwise} \end{cases}$$

The algorithm is a search algorithm where the elements of x are assigned values 0 or 1 and the elements that sum up to ℓ are checked in the process until all the combinations are generated. The algorithm is given in a recursive form below.

Let us suppose that the elements of x from i = 1 to k-1 have been assigned. Then we define s as :

$$s = \sum_{i=1}^{k-1} x_i$$

and r as :

$$r = \sum_{i=k}^{n} q_i$$

The the following recursive algorithm PARTS(s,k,r,ℓ) yields all partitions whose sum is ℓ

STEP 1 Generate left child of the tree i.e :

$x_k = 1$

If $s + q_k = \ell$ a subset whose elements sum up to ℓ is generated.

Else if $s + q_k + q_{k+1} < \ell$ then all elements up to k have been assigned thus we repeat the procedure by calling PARTS($s+q_k$,k+1,$r-q_k$,ℓ)

STEP 2 Generate right child of the tree i.e. :

$x_k = 0$

If $s + r - q_k > \ell$ and $s + q_k + 1 < \ell$ then all elements up to k have been assigned and we repeat the procedure by calling PARTS $(s,k+1,r-q_k,\ell)$.

The algorithm is initiated by calling PARTS$(0,1,\Sigma q_i,\ell)$.

To generate all the feasible subsets then we call the algorithm PARTS for each feasible $\ell \in F$. The algorithm has been tested on ten test problem which are taken from *EILON et al [47]*. Table 5-1 gives the description of the test problems that were used in the testing of the above algorithm for the generation of all feasible subtrees rooted at vertex v_0. These test problems are the same that will be used to solve the CST by means of a development of an algorithm that solves the CCSSP. Even though the algorithm is exponential, its performance is relatively good as we can see in Table 5-2. Table 5-2 reflects the results of using algorithm PARTS in generating all the feasible subsets rooted at vertex v_0

Problem Description				
#	n	M	Q	Source
1	11	4	24	Christofides et al [1981]
2	13	4	6000	Eilon et al [1971]
3	13	3	6500	Eilon et al [1971]
4	22	4	6000	Eilon et al [1971]
5	22	6	4000	Eilon et al [1971]
6	16	5	55	Eilon et al [1971]
7	16	3	90	Eilon et al [1971]
8	21	6	58	Eilon et al [1971]
9	21	4	85	Eilon et al [9171]
10	26	8	48	Eilon et al [1971]

Table 5-1 : Data Description of the test problems

Problem #	1	2	3	4	5	6	7	8	9	10
# of subtrees in T_0	40	538	442	54316	7859	515	2250	4170	16852	9056

Table 5-2 : Number of feasible subtrees in each problem.

Let us rewrite the relaxed formulation for the CST :

$$\min \sum_{i \in I} \sum_{\ell \in F} \tilde{d}_{i\ell} \xi_{i\ell} \tag{25}$$

subject to :

$$\sum_{\ell \in F} \xi_{i\ell} = 1 \qquad \forall i \in V \tag{26}$$

$$\sum_{i \in I} \sum_{\ell \in F} \left(\frac{q_i}{\ell}\right) \xi_{i\ell} = M \tag{27}$$

$$x_{i\ell} \in \{0,1\} \qquad \forall i \in V, \ \forall \ell \in F \tag{28}$$

where $\tilde{d}_{i\ell} = d_{i\ell} \times \left(\frac{q_i}{\ell}\right)$.and $d_{i\ell}$ is the value of the minimum cost spanning tree rooted at v_0 containing vertex i and of load ℓ.

Having now derived the values for $d_{i\ell}$ as mentioned above, we solve problem (26)-(28) by relaxing constraint (27) in a Lagrangean fashion. In this way the solution to the problem is straightforward and is given by :

$$\max_{\gamma} \left\{ \min \sum_{i \in I} \sum_{\ell \in F} \left(\tilde{d}_{i\ell} + \left(\frac{q_i}{\ell}\right)\gamma \right) \xi_{i\ell} - M\gamma \right\} \tag{29}$$

The solution is obtained, using subgradient optimization, by setting $\xi_{i\ell} = 1$ for that ℓ which yields the minimum of (29) and $\xi_{i\ell} = 0$ for the rest of $\ell \in F$. This finally leads

to a lower bound for the CST.

Table 5-3 gives the corresponding lower bounds to the ten test problems as well as the value of the MST and the computing time required to get those lower bounds inclusive of the generation process, the MST computation and the Lagrangean iterations.

Problem #	1	2	3	4	5	6	7	8	9	10
Value of SST	121	103	103	234	234	180	180	218	218	253
Lower Bound for the CMST at end of LR *	128	148	151	266	275	201	188	257	244	340
Time in CP seconds	.173	2.52	2.39	556.87	52.09	2.86	18.92	28.88	163.27	72.50

Table 5-3 : Computational results for the Lagrangean Relaxation.

* Value rounded to next integer.

5-4 REDUCTIONS TESTS

As we could see it from Table 5-2 some of the test problems have a number of variables that far exceeds the 4000 to 5000 limit for an SSP algorithm. Clearly it is not possible to use (1)-(4) to solve these problems because of the limitations of the CCSSP algorithm. Therefore, we will try to reduce the number of variables to a reasonable size so that we can use an algorithm to solve the CCSSP.

In other words we want to derive a subset $T_1 \subset T_0$ that is much smaller in cardinality than T_0 but is guaranteed to contain the optimal solution for problem (20)-(25).

This will be achieved by using some reductions tests.

Let us first give some graph theory definitions that are taken from *FULKERSON*

et al [53]:

V and A are defined as before on G. Let T be a spanning tree in G.

Definition 1 : An arc of G in T is called a branch and an arc of G not in T is called a chord.

Definition 2 :The fundamental path of a chord (p,q) is the unique chain of branches of T between p and q.

Definition 3 : For two disjoint subsets N_1 and N_2 the cutset of arcs of G is defined by $\{(i,j)/i \in N_1, j \in N_2\}$. The fundamental cutset of a branch (s,t) is the cutset of the two connected subgraphs of G that remain when (s,t) is removed from T.

A minimum spanning tree T is a spanning tree of minimum length L(T). Two equivalent necessary and sufficient conditions for a spanning tree to be minimal are :

(a) every branch of the tree is at least as short as any chord in its fundamental cutset.
(b) every chord of the tree is at least as long as any branch in its fundamental path.

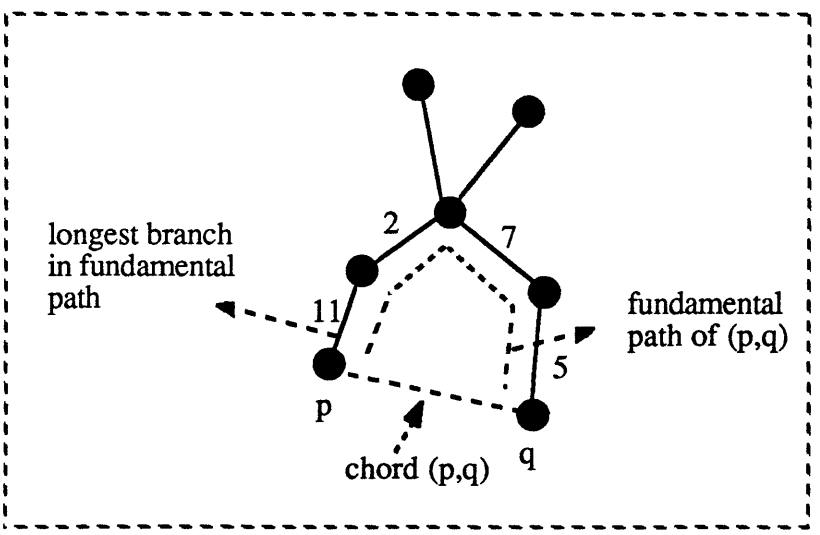


Figure 5-6 : Fundamental path of a chord (p,q).

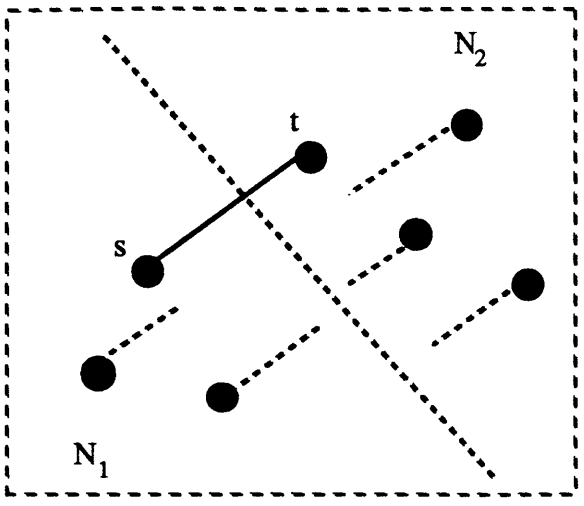


Figure 5-8 : Fundamental cutset of branch (s,t).

With these definitions we devise an analysis that will permit us to forbid arcs from belonging to any optimal solution for the CST and to require arcs to be part of any optimal solution for the CST.

Having a value u* as an upper-bound on the value of the optimal solution to the CST and T being the MST on G , we then can say that :

(a) a branch (a,b) is indispensable if

$$L(T_{ab}) = L(T) + C(a,b) + C(r_{ab},s_{ab}) > u^*$$

(b) a chord (p,q) is forbidden if :

$$L(T_{pq}) = L(T) + C(p,q) - C(r_{pq},s_{pq}) > u^*$$

where T_{ab} is the tree that arises from T after fixing chord (a,b) in, and T_{ab} is the tree that arises from T after fixing branch (a,b) out.

L(T) is the value cost of the tree T.

We can outline the analysis as follows :

(1) Let u* be an upper bound on the optimal value of the CST and T be the MST on graph G :

a branch (a,b) is indispensable if $L(T_{ab}) > u^*$

a chord (p,q) is forbidden if $L(T_{pq}) > u^*$

(2) At the same time in the case of a required arc we check that if the only way for

this arc to remain connected to the rest of arcs is through a chain of other arcs that are unique then we make those arcs required too.

(3) Repeat (1) and (2) until no further improvement is possible.

Reduction test :

With the forbidden arcs derived in the analysis above one then excludes any subset that sum up to ℓ during the generation process using algorithm PARTS if the MST in that subset includes any of the arcs forbidden. Also, when we generate a feasible set, we can compute the MST in that set and the MST in the remaining set i.e if S is the set generated then V-S is the complement of S in V. If the sum of the values of the two MST exceeds a known upper-bound on the problem then we eliminate from further consideration that subset. Also a subset is excluded if a forbidden arc belongs to the MST in that subset.

We can then outline a procedure that will enable us to solve the CST problem as follows:

STEP 1 Use the analysis described above to forbid arcs belonging to any optimal solution of the CST.

STEP 2 Generate all the combinations of the q_i that sum up to ℓ for every $\ell \in F$ using algorithm PARTS. During the process, in each subset, derive the corresponding MST then exclude that subset if :

(a) Any of the arcs forbidden belong to the MST of that subset.

(b) The sum of the value of the MST in that subset plus the value of the MST in the complement of that subset is greater than an upper bound to the optimal solution.

Example :

Let us consider the following graph of 7 vertices

0	28	21	14	17	18	22
28	0	47	36	25	20	35
21	47	0	26	37	30	20
14	36	26	0	15	31	34
17	25	37	15	0	29	39
18	20	30	31	29	0	16
22	35	20	34	39	16	0

Table 5-5 : Distance matrix

vertex	1	2	3	4	5	6
Load	2	3	1	1	2	1

Table 5-6 : Vertex loads q_i i=1,...,6.

If we want to cluster this network into two subsets i.e. M=2 then we want to generate all feasible subsets of load less or equal to Q=6 (that is the maximum capacity load for each subset).

The feasible loads are 4,5,and 6 because 6 is an upper-bound and the lower bound is equal to $(\Sigma q_i - (M-1)Q) = 10-6 = 4$.

Let us apply the procedure described above for the reduction of the subsets.

STEP 1 The analysis described above is used on the MST of graph G defined by Table 5-5

The minimum spanning tree T is as follows :

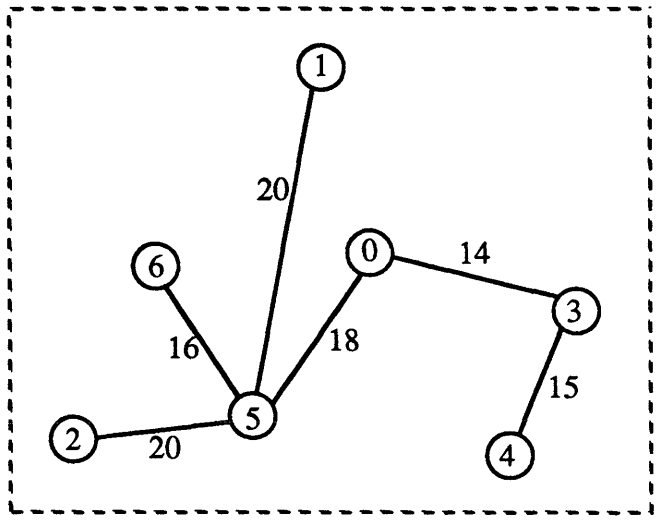


Figure 5-2 : MST for the data set in Table 5-5

L(T) = 103 this tree is not feasible for the CST problem.

A feasible tree for the CST is shown below.:

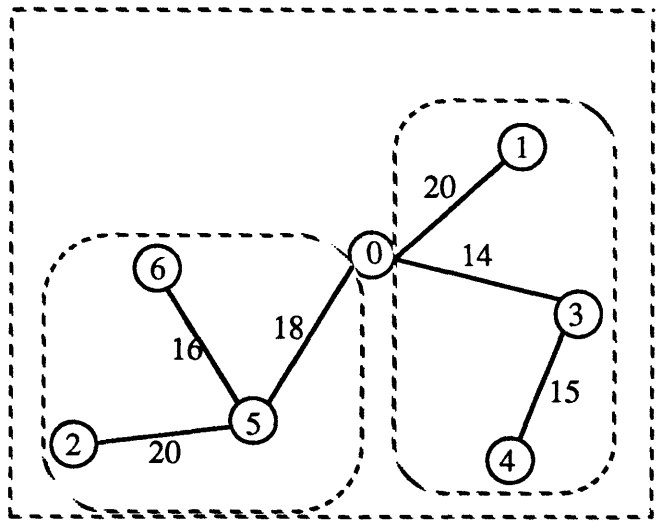


Figure 5-3 : A feasible tree for the CST.

Using the analysis we get :

Forbidden arcs :

(0,1) , (1,2) , (2,3) , (3,5) , (4,5)

(1,3) , (2,4) , (3,6) , (4,6)

(1,4) , (2,5)

(1,6)

Required arcs :

(1,5)

We can improve on this by noticing that since arc (1,5) is required and that it should remain connected in any optimal solution to the CST then arcs (0,5) and (5,6) are required because (2,3),(2,4),(3,6),(4,6) are forbidden. From this we get also that (2,6) should be forbidden.

Using the information above we get the following tree which is in fact the optimal solution to the CST :

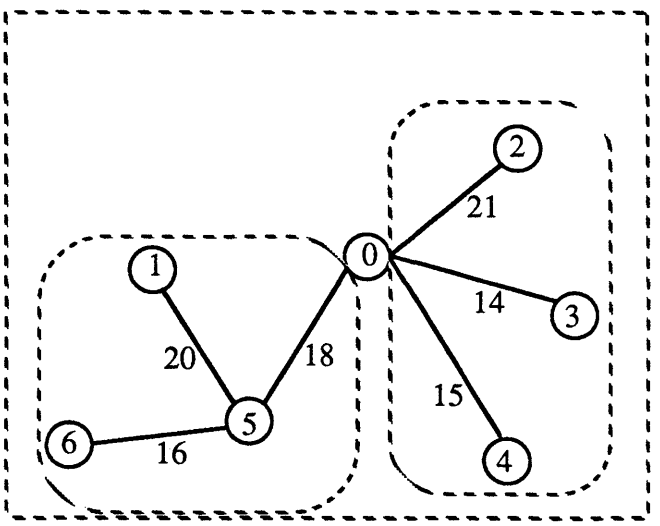


Figure 5-4 : Optimal tree for the CST example.

L(T) is equal to 104

STEP 2 Generating all feasible subsets we find that there are 30 of them namely the following :

Sets of load 4 :

1) Set {0,1,3,4} with MST cost equal to 54

remaining vertices have MST cost equal to 54

2) Set {0,3,4,5} with MST cost equal to 47
remaining vertices have MST cost equal to 69

3) Set {0,1,3,6} with MST cost equal to 64
remaining vertices have MST cost equal to 56

4)Set {0,3,5,6} with MST cost equal to 48
remaining vertices have MST cost equal to 63

5) Set {0,2,3} with MST cost equal to 35
remaining vertices have MST cost equal to 71

6)Set {0,1,4,6} with MST cost equal to 64
remaining vertices have MST cost equal to 53

7) Set {0,4,5,6} with MST cost equal to 51
remaining vertices have MST cost equal to 63

8) Set {0,2,4} with MST cost equal to 38
remaining vertices have MST cost equal to 68

9) Set {0,2,6} with MST cost equal to 41
remaining vertices have MST cost equal to 67

10) Set {0,1,5} with MST cost equal to 38
remaining vertices have MST cost equal to 70

Sets of load 5 :

11) Set {0,1,3,4,6} with MST cost equal to 76
remaining vertices have MST cost equal to 39

12) Set {0,3,4,5,6} with MST cost equal to 63
remaining vertices have MST cost equal to 49

13) Set {0,2,3,4} with MST cost equal to 50
remaining vertices have MST cost equal to 54

14) Set {0,2,3,6} with MST cost equal to 55
remaining vertices have MST cost equal to 55

15) Set {0,1,3,5} with MST cost equal to 52

remaining vertices have MST cost equal to 58

16) Set {0,2,4,6} with MST cost equal to 58

remaining vertices have MST cost equal to 52

17) Set {0,1,4,5} with MST cost equal to 55

remaining vertices have MST cost equal to 55

18) Set {0,1,5,6} with MST cost equal to 54

remaining vertices have MST cost equal to 50

19) Set {0,1,2} with MST cost equal to 49

remaining vertices have MST cost equal to 63

20) Set {0,2,5} with MST cost equal to 39

remaining vertices have MST cost equal to 76

Sets of load 6 :

21) Set {0,2,3,4,6} with MST cost equal to 70

remaining vertices have MST cost equal to 38

22) Set {0,1,3,4,5} with MST cost equal to 67

remaining vertices have MST cost equal to 41

23) Set {0,1,3,5,6} with MST cost equal to 68

remaining vertices have MST cost equal to 38

24) Set {0,1,2,3} with MST cost equal to 63

remaining vertices have MST cost equal to 51

25) Set {0,2,3,5} with MST cost equal to 53

remaining vertices have MST cost equal to 64

26) Set {0,1,4,5,6} with MST cost equal to 71

remaining vertices have MST cost equal to 35

27) Set {0,1,2,4} with MST cost equal to 63

remaining vertices have MST cost equal to 48

28) Set {0,12,4,5} with MST cost equal to 56

remaining vertices have MST cost equal to 64

29) Set {0,1,2,6} with MST cost equal to 69

remaining vertices have MST cost equal to 47

30) Set {0,2,5,6} with MST cost equal to 54

remaining vertices have MST cost equal to 54

In fact just the second part of the test is enough to eliminate all the subsets except subsets {0,1,5,6} and {0,2,3,4} which correspond to the optimal solution for the CST. Also one should have noticed that in fact in this particular example the analysis was enough to derive the optimal solution for the problem.

We have tested the above procedure on the ten test problems in Table 5-1 and could effectively reduce the number of feasible subtrees that can be taken into consideration when trying to solve the CCSSP to optimality.

Table 5-7 gives all the detail of the running time of the analysis and the reductions made in terms of subsets eliminated so that the number of subsets left is small enough to be used in a exact method solution to be solved.

Problem	1	2	3	4	5	6	7	8	9	10
# of arcs in graph	55	78	78	231	231	120	120	210	210	325
# of arcs eliminated	44	54	56	189	197	85	90	172	177	260
# of arcs required	2	2	3	1	0	1	0	0	0	1
# of initial subsets	40	538	442	54316	7859	515	2250	4170	16852	9056
# of subset remaining after analysis	8	121	112	16186	2150	75	890	1120	4415	3120

Table 5-7 : Reductions of the number of variables for the CCSSP

5-5 TREE SEARCH FOR THE CCSSP

To get solutions to the CST using formulation (1)-(4) one needs to have a subset $T_1 \subset T_0$ that is much smaller in cardinality than T_0. On the other hand formulation (1)-(4) which has been denoted as CCSSP can be solved optimally by transforming it to an equivalent formulation which will yield lower bounds to (1)-(4) as discussed in previous sections. In section 5-3-2 we showed that :

$$\max_{\gamma}\left\{ \min \sum_{i\in I}\sum_{\ell\in F}\left(\tilde{d}_{i\ell} + \left(\frac{q_i}{\ell}\right)\gamma\right)\xi_{i\ell} - M\gamma \right\} \qquad (30)$$

will give us a lower bound on formulation (1)-(4).

Since we are taking a small subset T_1 instead of T_0 we can generate all the $d_{i\ell}$ and use in them in (30) to get a lower bound on problem (1)-(4) that will be embedded in a tree search. The $d_{i\ell}$ are generated by taking :

$$d_{i\ell} = \min_{t\in T_1}\left[d_t \bullet \left(\frac{q_i}{\ell}\right)\right]. \qquad \forall\, i \in V\,;\, \forall\, \ell \in F.$$

Lower bounds are derived at every node of the tree by performing a fixed number of iterations of the Subgradient Optimization technique in relation with (30). A depth first search strategy was followed where a tree $t \in T_1$ is forced to be in the optimal solution until either we hit on a new feasible solution of the problem that is better than the upper bound that we have at hand, or the lower bound associated with the node being investigated exceeds the best upper bound. When the latter happens one backtracks to the parent node being investigated and forces that tree t to be out of the solution. The search is repeated until no more node can be investigated and the optimal solution will be the best feasible solution recorded. The branching strategy was to branch on the tree $t \in T_1$ that appears most in the solution of (30). The upper bound to the problems was derived using the heuristic of Esau-Williams that was

discussed in Chapter 1. This heuristic performed well.

The description of problem 7 is given in table 5-1 and after the reduction tests the CCSSP consisted of 890 subsets that were left. These subsets have been numbered from 1 to 890 and Figure 5-8 gives the tree search of the CCSSP algorithm which consisted in generating 13 nodes after which the optimal solution was found.

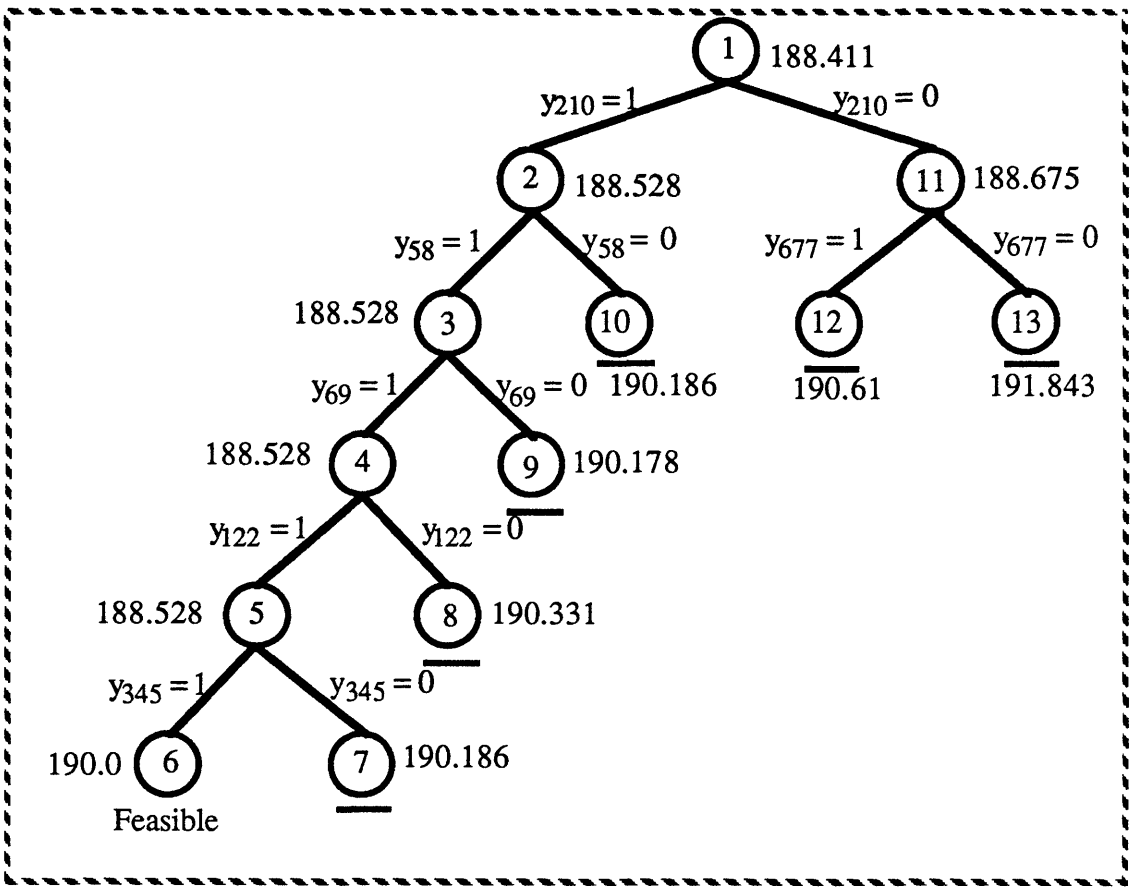


Figure 5-8 : Tree search for test problem 7.

"___" means has been fathomed.

Table 5-7 gives the results of the algorithm applied to the ten test problems where one can see that all the problems were solved to optimality except for test problem 4. This is due to the fact that the set T_1 (which consists of the subtrees that were not eliminated by the reduction test) was too big in cardinality to enable the CCSSP algorithm to be used efficiently in this case. The algorithm was programmed in PASCAL and tested on a SUN 3-60 workstation. The speed of a SUN 3-60

worksation is approximately half of that of the Cyber 855.

Problem #	# of nodes in tree search	Upperbound	Bound at root node	Value of optimal solution	Execution time (secs)
1	0	136	131.0	131	0.1
2	5	160	141.878	155	5.6
3	1	175	136.725	160	0.15
4	**	**	**	**	**
5	130	295	284.406	286	314.2
6	91	206	201.624	202	17.3
7	13	203	188.411	195	48.2
8	546	281	261.534	267	206.1
9	231	264	248.345	250	412.3
10	1028	372	353.749	360	850.7

Table 5.7 : Results of the CCSSP algorithm after tests reductions

5-6 CONCLUSIONS

In this Chapter we have introduced a formulation for the CST which is based on a Set Partitioning Algorithm formulation. We have to chose among a enormous number of subtrees a very small subset that satisfy the capacity load and spanning tree condition. This has been done by trying to eliminate as many subtrees as possible and then solve a Cardinality Constrained Set Partitioning Problem. The

difficulties arise when we need good reductions tests in order to make the elimination process a successful one. Some basic tests have been used and we hope that some new and stronger tests will be devised in the future to make a better use of this formulation.

CHAPTER 6

CONCLUSIONS

We have considered in this thesis exact solution approaches for the Capacitated Spanning Tree problem (CST). The literature survey of the methods used to solve the CST showed that very little work has been done to find exact algorithms for the CST. In fact for this particular problem and given its complexity, researchers were mainly interested in deriving good feasible solutions to the problem. So the interest was greatly heightened in trying to derive new formulations for the CST in order to be able to solve it exactly. This was done by formulating the problem in two different ways. The first one was a 0-1 integer programming formulation in two sets of variables that was well suited for the use of Benders' Decomposition scheme. The interest in this method is that very few researchers achieved good results in using it. The main drawback for this method is that it needs the solution of an integer programming problem for each iteration. As we know the solution of an integer program is in itself a hard problem and very costly in terms of

computing time. Therefore, the design of good formulations plays a key role in limiting the number of iterations that one has to perform in order to achieve optimality. Fortunately with our formulation this proved to be the case where in most situations the algorithm needed few iterations to terminate optimally.

The second part of the thesis was the derivation of a second formulation that is based on another 0-1 integer programming formulation in one set of variables. This formulation shows well the combinatorial nature of the problem since it is based on set partitioning ideas. Unfortunately this formulation uses a huge number of variables. Some of the ideas developed in this second part of the thesis to solve this formulation stem from the work of previous researchers working on other combinatorial problems. They aim is to reduce the number of variables in order to make the problem easier to solve by means of solving a Cardinality Constrained Set Partitioning Problem. Various reduction tests were devised in order to reduce this number of variables and the procedure worked well in all but one of the cases that we tested.

The exact algorithms developed in this thesis enable the optimal solution of the Capacited Spanning Tree problem for sizes up to approximately 30 vertices in reasonable computing time.

REFERENCES

1. *Bahl H.C. and Zionts S. [1987]*, "Multi-Item Scheduling by Benders' Decomposition", J. of Operational Research Soc., Vol 38, No 12, pp. 1141-1148.

2. *Balas E. [1965]*, "An additive Algorithm for solving Linear Programs with Zero-One variables", Operations Research, Vol 13, pp 517-546.

3. *Balas E. and Bergthaller E.[1976]*, "Benders' Method Revisited", Management Science Report No 401, Graduate School of Industrial Administration, Carnegie-Mellon University, U.S.A.

4. *Balas E. and Padberg M. [1979]*, "Set Partitioning - a survey", In Combinatorial Optimization, N. Christofides, A. Mingozzi, P. Toth and C. Sandi (Editors).

5. *Balas E. and Christofides N. [1981]*, "A Restricted Lagrangean Approach for the Traveling Salesman Problem", Mathematical Programming, Vol 21, pp 19-46.

6. *Balinski M. [1965]*, "Integer Programming : Methods, Uses , Computations", Management Science, Vol 12, No 3, pp 253-313.

7. *Balinski M. and Quandt R. [1964]*, "On an Integer Program For the Delivery Problem", Operations Research, Vol 12, pp 300-312.

8. *Bazaraa M.S. and Jarvis J.J. [1977]*, "Linear Programming and Network

Flows", John Wiley & Sons, New York, U.S.A.

9. *Bazaraa M.S. and Goode J.J. [1979]*, "A survey of Various Tactics for Generating Lagrangean Multipliers in the Context of Lagrangean Duality", European Journal of Operational Research, Vol 3, pp 322-338.

10. *Bazaraa M.S. and Shetty C.M. [1979]*, "Nonlinear Programming : Theory and Algorithms", John Wiley & Sons, New York, U.S.A.

11. *Beale E.M.L. [1965]*, "Survey of Integer Programming", Operational Research Quarterly, Vol 16, pp 219-228.

12. *Beale E.M.L. [1979]*, "Branch and Bound Methods for Mathematical Programming Systems", Annals of Discrete Mathematics, Vol 5 : Discrete Optimization II, pp 201-219

13. *Beasley J. [1983]*, " A New Algorithm For The Steiner Problem", Networks, Vol 14, pp 147-160.

14. *Beasley J. [1983]*, "Route First-Cluster Second Methods for Vehicle Routing", Omega, Vol 11, pp 403-408.

15. *Bellman R. [1957]*, "Dynamic Programming", Princeton University Press, Princeton, N.J., U.S.A.

16. *Bellman R. and Dreyfus S.E. [1957]*, "Applied Dynamic Programming", Princeton University Press, N.J., U.S.A.

17. *Bellmore M. and Malone J.C. [1971]*, "Pathology of Traveling Salesman

Subtour-Elimination Algorithms",Oper. Res., Vol 19, pp 278-307.

18. *Benders J. [1962]*, "Partitioning Procedures for Solving Mixed Variable Programming Problems", Numerische Mathematic, Vol 4, pp 238-252.

19. *Berge C. [1970]*, "Graphes et Hypergraphes", Dunod, Paris.

20. *Bloom J.A., Caramanis M. and Charny L. [1984]*, "Long-Range Generation Planning Using Generalized Benders' Decomposition : Implementation and Experience", Operations Research, Vol 32, No 2, pp 290-313.

21. *Bock F. [1971]*, "An Algorithm to Construct a Minimum Directed Spanning Tree in a Directed Network", Developments in Operations Research, Gordon and Breach, pp. 29-44 ,New York, U.S.A.

22. *Boorstyn R. and Frank H. [1977]*, "Large-Scale Topological Optimization", IEEE Transactions on Communication, Vol COM-25, No 1, pp 29-47.

23. *Camerini P.M., Fratta L. and Maffioli F. [1979]*, "A Note on Finding Optimum Branchings", Networks, Vol 9, pp 309-312.

24. *Chalmet L.G. and Gelders L.F. [1976]*, "Lagrangean Relaxations for a Generalized Assignment-Type Problem", Katholieke Universiteit Leuven, Working Paper, No 76-121.

25. *Chandy K.M. and Russel R.A. [1972]*, "The Design of Multipoint Linkages in a Teleprocessing Tree Network", IEEE Transactions on Computers, Vol C-21, pp 1062-1066.

26. *Chandy K.M. and Lo T. [1973]*, "The Capacitated Minimum Spanning Tree", Networks, Vol 3, pp 173-181.

27. *Cheddad H. [1987]*, "Algorithms For Crew Scheduling Problems", Ph.D. thesis, Imperial College of Science and Technology, London.

28. *Cheriton D. and Tarjan R.E. [1976]*, "Finding Minimum Spanning Trees", SIAM J. of Computing, Vol 5, No 4, pp 724-742.

29. *Christofides N. [1975]*, "Graph Theory, an Algorithmic Approach", Academic Press, London.

30. *Christofides N. [1975]*, "Vehicle Routing", RAIRO Rech. Oper., Vol 10, pp 55-70.

31. *Christofides N. [1979]*, "The Traveling Salesman Problem", In Combinatorial Optimization (edited by N. Christofides, A. Mingozzi, P. Toth and C. Sandi), John Wiley & Sons.

32. *Christofides N., Mingozzi A. and Toth P. [1981]*, "State-Space Relaxation Procedures for the Computation of Bounds to Routing Problems", Networks, Vol 11, pp 145-164.

33. *Christofides N., Mingozzi A. and Toth P. [1981]*, "Exact Algorithms For The Vehicle Routing Problem Based On Spanning Trees And Shortest Path Relaxations", Mathematical Programming, Vol 20, pp 255-265.

34. *Christofides N. and Paixao J. [1982]*, "State-Space Relaxation Algorithms for

the Set-Covering Problem", Imperial College Report No IC,OR,81/12.

35. *Christofides N. [1985]*, "Vehicle Routing", In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnoy Kan, D.B. Shmoys (eds), The Traveling Salesman Problem, Wiley, Chichester, Ch 12.

36. *Chu Y. and Liu T.H. [1965]*, "On the Shortest Arborescence of a Directed Graph", Sci. Sinica, Vol 14, pp 1396-1400.

37. *Clarke G. and Wright J. (1959)*, " Scheduling of Vehicles from a Central Depot to a Number of Delivery Points", Oper. Res. 12, pp 568-581.

38. *Clasen R.J. [1983]*, "The Solution of The Chemical Equilibrium Programming Problem with Generalized Benders Decomposition", Operations Research, Vol 32, No 1, pp 70-79.

39. *Cook S.A. [1971]*, "The complexity of Theorem Proving Procedures", Proc. 3rd Ann. ACM Symp. on Theory of Computing, Association for Computing Machinery, New York, pp 151-158.

40. *Crowder H., Johnson E.L. and Padberg M. [1983]*, "Solving Large-Scale Zero-One Linear Programming Problems", Operations Research, Vol 31, No 5, pp 803-834.

41. *Dakin R.J. [1965]*, "A tree Search Algorithm for Mixed-Integer Programming Problems", Computer Journal, Vol 8, pp 250-255.

42. *Dantzig G. [1963]*, Linear Programming and Extensions, Princeton University Press, Princeton, N.J., U.S.A.

43. *Dantzig G. and Wolfe P. [1961]*, "The Decomposition Algorithm for Linear Programming", Econometrica, Vol 9, No 4.

44. *Deo N. [1974]*, "Graph Theory with Applications to Engineering and Computer Science", Prentice-Hall.

45. *Dreyfus S.E. and Law A.M. [1971]*, "The Art and Theory of Dynamic Programming", Academic Press, New York, U.S.A.

46. *Edmonds J.[1967]*, "Optimum Branchings", Jour. of Research of the National Bureau of Standards, Vol 71B, pp 233-240.

47. *Elias D. and Ferguson M.J. [1974]*, "Topological Design of Multipoint Teleprocessing Networks", IEEE Trans. on Com., Vol COM-22, pp 1753-1761.

48. *Esau L.R. and Williams K.C.[1966]*, " On Teleprocessing System Design , Part II", IBM Syst. J., Vol 5, No 3 , pp 142-147.

49. *Eilon S., Watson-Gandy C.D.T. and Christofides N. [1971]*, Distribution Management : Mathematical Modelling and Practical Analysis, Griffin, London.

50. *Fisher M.L. and Jaikumar R. [1978]*, "A Decomposition Algorithm for Large-Scale Vehicle Routing", Report 78-11-05, Department of Decision Sciences, The Wharton School , University of Pennsylvania, Philadelphia, U.S.A.

51. *Fisher M.L. [1981]*, "The Lagrangean Relaxation Method for Solving Integer Programming Problems", Management Science, Vol 27, No 1, pp 1-18.

52. *Fisher M. and Jaikumar R. [1981]*, "A Generalized Heuristic for Vehicle

Routing", Networks, Vol 11, pp 111-124.

53. *Fisher M.L. [1985]*, "An Application Oriented Guide to Lagrangean Relaxation", Interfaces, Vol 15, No 2, pp 10-21.

54. *Florian M., Bushell G., Ferland J., Guerni G. and Nastansky L. [1976]*, "The Engine Scheduling Problem in a Railway Network", INFOR , Vol 14, No 2, pp 121-138.

55. *Ford L. and Fulkerson D. [1962]*, "Flows in Networks", Princeton University Press, Princeton, New Jersey, U.S.A.

56. *Fox B.L. [1978]*, "Data Structures and Computer Science Techniques in Operations Research", Operations Research, Vol 26, No 5, pp 686-717.

57. *Franca P.M. and Luna H.P.L.[1982]*, " Solving Stochastic Transportation-Location Problems by Generalized Benders Decomposition", Transportation Science, Vol 16, No 2, pp 113-126.

58. *Frank H., Frisch, Van Slyke R. and Chou W.S. [1971]*, "Optimal Design of Centralized Computer Networks", Networks, Vol 1, pp 43-57.

59. *Gabow H.N. [1977]*, "Two Algorithms For Generating Weighted Spanning Trees in Order", SIAM Jour. of Computing ,Vol 6, pp. 139-150.

60. *Gabow H.N. [1978]*, " A Good Algorithm for Smallest Spanning Trees with a Degree Constraint", Networks, Vol 8, pp 201-208.

61. *Gallo G. and Pallotino S. [1986]*, "Shortest Path Methods, a Unifying Approach", Mathematical Programming Study, Vol 26, pp 38-64.

62. *Garey M.R. and Johnson D.S. [1979]*, "Computers and Intractability: A Guide to the Theory of NP-Completeness", Freeman &Co, San Fransico, U.S.A.

63. *Garfinkel R.S. [1979]*, "Branch and Bound Methods for Integer Programming", in Combinatorial Optimization, Christofides N., Mingozzi A., Toth P. and Sandi C. editors, John Wiley & Sons, pp 1-20.

64. *Garfinkel R.S. and Nemhauser G.L. [1972]*, "Integer Programming", John Wiley & Sons.

65. *Gavish B. [1978]*, "On Obtaining the "Best" Multipliers for Lagrangean Relaxation for Integer Programming", Computers and Operations Research, Vol 3, pp 322-338.

66. *Gavish B [1982]*, "Topological Design of Centralized Networks - Formulations and Algorithms", Networks, Vol 12, pp 355-377.

67. *Gavish B. [1983]*, "Formulations and Algorithms for the Capacitated Minimal Directed Tree Problem", J. of ACM, Vol 30, No 1, pp 118-132.

68. *Gavish B. [1985]*, " Augmented Lagrangean Relaxation Based Algorithms for Centralized Network Design", IEEE Transactions on Communication, Vol COM-33, pp 1247-1257.

69. *Geoffrion A.M. [1967]*, "Integer Programming by Implicit Enumeration and Balas' Method", SIAM Review, Vol 9, pp 178-190.

70. *Geoffrion A.M. and Marsten R.E. [1972]*, "Integer Programming Algorithms : A Framework and State-of-the-Art Survey", Management Science, Vol 18, No 9, pp 465-491.

71. *Geoffrion A.M. [1972]*, "Generalized Benders Decomposition", J. of Optimization and Applications, Vol 10, No 4, pp 237-260.

72. *Geoffrion A.M. [1974]*, "Lagrangean Relaxation for Integer Programming", Mathematical Programming Study, Vol 2, pp 82-114.

73. *Geoffrion A.M. and Graves G. [1974]*, "Multicommodity Distribution System Design by Benders Decomposition", Management Science, Vol 20, pp 822-844.

74. *Geoffrion A.M. and Mac Bride R. [1978]*, "Lagrangean Relaxation Applied to Capacitated Facility Location Problems", AIIE transactions, Vol 10, pp 40-47.

75. *Gillet and Miller [1974]*, "A Heuristic Algorithm for the Vehicle-Dispatch Problem", Oper.Res. 22, pp. 340-349.

76. *Gomory R.E. [1958]*, "An Algorithm for the Mixed Integer Problem", RM-2597, RAND Corporation.

77. *Gomory R.E. [1963]*, "An Algorithm for Integer Solutions to Linear Programs", In Graves and Wolfe (eds), Recent Advances in Mathematical Programming, pp 269-302.

78. *Harary F. [1971]*, "Graph Theory", Adison-Wesley.

79. *Hayes R.L. [1967]*, "The Delivery Problem", Carnegie Institute of Technology, Graduate School of Administration, Pittsburg, Report No MSR 106.

80. *Held M. and Karp R.M. [1970]*, "The Traveling Salesman Problem and Minimum Spanning Trees", Operations Research, Vol 18, pp 1138-1162.

81. *Held M. and Karp R.M. [1971]*. "The Traveling Salesman Problem and Minimum Spanning Trees : Part II", Mathematical Programming, Vol 1, pp 6-25.

82. *Held M. , Wolfe P. and Crowder H.P. [1974]*, "Validation of Subgradient Optimization", Mathematical Programming, Vol 6, pp 62-88.

83. *Hoffman A.J. and Kruskal J.B. [1956]*, "Integral Boundary Points of Convex Polyhedra", in H.W.Kuhn and A.W. Tucker (eds), Linear Inequalities and Related Systems, Annals of Mathematics Studies, No 38, Princeton, New Jersey, U.S.A.

84. *Horowitz E. and Sahni S. [1974]*, "Computing Partitions with Applications to the Knapsack Problem", SIAM Jour. of Computing, Vol 21, pp 277-288.

85. *Horowitz E. and Sahni S. [1978]*, "Fundamentals of Computer Algorithms", Pitman.

86. *Johnson E. [1978]*, Lectures Notes, Combinatorial Optimization Summer School, Urbino , Italy.

87. *Karp R.M. [1972]*, "A Simple Derivation of Edmonds' Algorithm for Optimum Branchings", Networks , Vol 1, pp. 265-272.

88. *Karp R.M. [1972]*, "Reducibility among Combinatorial Problems", in R.E. Millet and J.W. Thatcher (eds), Complexity of Computer Computations, Plenum Press, New York, pp 85-103.

89. *Karp R.M. [1975]*, "On the Complexity of Combinatorial Problems", Networks, Vol 5, pp 45-68.

90. *Karnaugh M.[1972]*, "Multipoint Network Layout Problem", International Business Machine Corp., Report # RC 3723.

91. *Katoh N., Ibaraki T. and Mine H. [1981]*, "An Algorithm For Finding K Minimum Spanning Trees", SIAM Jour. of Computing, Vol 10 , No2, pp 247-255.

92. *Kershenbaum A. and Van Slyke R. [1972]*, "Computing Minimum Spanning Trees Efficiently", Proc. ACM ANNUAL CONF., pp 518-527.

93. *Kershenbaum A. and Chou A.W. [1972]*, "A unified Algorithm For Designing Multidrop Teleprocessing Networks", IEEE Transactions on Communication, Vol COM-22, pp 1762-1772.

94. *Kershenbaum A. [1976]*, "Centralized Teleprocessing Network Design", Ph.D. Thesis, Polytechnic Institute of New York, U.S.A.

95. *Kershenbaum A. and R. Boorstyn [1980]*, "Second-Order Greedy Algorithm for Centralized Teleprocessing Network Design", IEEE Transactions on Communications, Vol COM-28, No 10, pp 1835-1843.

96. *Kirpatrick S., Gelat J.C. and Vecchi M.P. [1983]*, "Optimization By

Simulated Annealing", Science 220 , pp 671-680.

97. *Klingman D. [1977]*, "Finding Equivalent Network Formulations for Constrained Network Formulations", Management Science, Vol 23, No 7.

98. *Klingman D. and Russel R. [1975]*, "Solving Constrained Transportation Problems", Operations Research, Vol 23, No 1.

99. *Kobayashi H. [1982]*, " Communication Network Design and Control Algorithms - A Survey", RC 9233 Computer Science IBM.

100. *Kruskal J.B. [1956]*, "On The Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem", Proc. Amer. Math. Soc., Vol 7.

101. *Land A. and Doig A.G. [1960]*, "An Automatic Method for Solving Discrete Programming Problems", Econometrica, Vol 28, pp 497-520.

102. *Land A. and Powell S. [1960]*, "Computer Codes for Problems of Integer Programming", Presented at the Conference on Discrete Optimization, Vancouver, B.C.

103. *Lawler E.L. and Wood D.E. [1960]*, "Branch-and-Bound Methods : A survey", Operations Research, Vol 14, pp 699-719.

104. *Lawler E.L. [1976]*, Combinatorial Optimization : Networks and Matroids, Holt , Rinehart and Winston, New York, U.S.A.

105. *Ladson L.S. [1970]*, "Optimization Theory for Large Scale Systems", The Macmillan Company.

106. *Lenstra J.K. and Rinnoy Kan A.H.G. [1979]*, "Computational Complexity of Discrete Optimization Problems", Annals of Discrete Mathematics, Vol 4 : Discrete Optimization I, pp 121-140.

107. *Lin S. and Kernighan W. [1971]*, "An effective Heuristic Algorithm for the Traveling Salesman Problem", Operations Research, Vol 10, pp 498-516.

108. *Lin S. [1975]*, "Computer Solution of the TSP", Bell System Tech. J., Vol 44, pp 2245.

109. *Little J.D.C., Murty K.C., Sweeney D.W. and Karel C. [1976]*, "An Algorithm for The Traveling Salesman Problem", Operations Research, Vol 11, pp. 972-989.

110. *Lucena A. [1986]*, Algorithms for the Vehicle Routing Problem", Ph.D. Thesis, Department of Management Science, Imperial College, London.

111. *Maffioli F. [1979]*, "The Complexity of Combinatorial Algorithms and the Challenge of Heuristics", Combinatorial Optimization, Christofides N., Mingozzi A., Toth P. and Sandi C. editors, John Wiley & Sons, pp 107-129.

112. *Magnanti T.L. [1981]*, "Accelerating Benders Decomposition : Algorithmic Enhancement and Model Selection Criteria", Operations Research, Vol 29, No 3, pp 464-484.

113. *Magnanti T.L. and Wong R.T. [1981]*, "Accelerating Benders' Decomposition : Algorithmic Enhancement and Model Selection Criteria", Operations Research, Vol 29, pp 464-484.

114. *Marsten R.E. [1974]*, "An Algorithm for Large Set-Partitioning Problems", Management Science, Vol 20, pp 774-787.

115. *Marsten R.E. [1981]*, "The Design of the XMP Linear Programming Library", ACM Transactions on Mathematical Software, Vol 7, No 4, pp 481-497.

116. *Martello S. and Toth P. [1978]*, "Algorithm for the Solution of the 0-1 Single Knapsack Problem",Computing, Vol 21, pp 81-86.

117. *Martello S.and Toth P. [1981]*, "An Algorithm for the Generalized Assignment Problem", Operational Research, pp 589-603.

118. *Martin J. [1972]*, System Analysis for Data Transmission. , Engewood Cliffs , New Jersey, Prentice-Hall, U.S.A.

119. *McDaniel D. and Devine M. [1977]*, "A Modified Partitioning Algorithm for Mixed Integer Programming", Management Science, Vol 24, No 3, pp 312-319.

120. *Mole R., Johnson D. and Weels K. [1983]*, "Combinatorial Analysis of Route First-Cluster Second Vehicle Routing", Omega, Vol 11, pp 507-512.

121. *Nemhauser G.L. [1966]*, "Introduction to Dynamic Programming", John Wiley & Sons, New York, U.S.A.

122. *Noonan F. and Giglio R.J. [1977]*, "Planning Electric Power Generation : A Nonlinear Mixed Integer Model Employing Benders Decomposition", Management Science, Vol 23, No 9, pp 946-956.

123. *Paixao J. [1983]*, "Algorithms for Large Scale Set Covering Problems",

Ph.D. Thesis, Department of Management Science, Imperial College, London.

124. *Papadimitriou C.H. [1978]*, "The complexity of the Capacitated Tree Problem", Networks, Vol 8, pp 217-230.

125. *Prim R.C. [1957]*, "Shortest Connection Networks and Some Generalizations", Bell Syst. Tech. J., Vol 36, pp 1389-1401.

126. *Riardin R.L. and Lin B.W. [1978]*, "What Makes Integer Programming Hard to Solve : An Empirical Study", Operations Research, Vol 15, Nos 2 and 3, pp 65-77.

127. *Richardson R. [1973]*, "Benders' Decomposition Method Applied to an Airline Routing Problem", Ph.D. dissertation , Department of Industrial Engineering, Systems Management Engineering and Operations Research, University of Pittsburgh, U.S.A.

128. *Ross G.T. and Soland R.M. [1975]*, "A Branch and Bound Algorithm for the Generalized Assignment Problem", Mathematical Programming, Vol 8, pp 91-103.

129. *Rubin D. [1973]*, "A hybrid Cutting-Planes Enumeration Algorithm for Integer Programming", 43rd National Meeting of ORSA, Milwaukee, May 11th.

130. *Sandi C. [1979]*, "Subgradient Optimization", In Combinatorial Optimization, N.Christofides, A. Mingozzi, P.Toth and C.Sandi, (Editors).

131. *Shapiro J.F. [1968]*, "Group Theoretic Algorithms for the Integer Programming Problem : Extension to a General Algorithm", Operations Research,

Vol 16, pp 928-947.

132. *Shapiro J.F.[1979]*, "A survey of Lagrangean Techniques of Discrete Optimization", Annals of Discrete Mathematics, Vol 5 : Discrete Optimization II, pp 113-138.

133. *Sharma R.L. and El-Bardai M.I. [1970]*, "Suboptimal Communications Network Synthesis", Proc. International Conf. on COMM, pp 19.11-19.16.

134. *Sherali H.D. [1981]*, "Expedients for Solving Some Specially Structured Mixed-Integer Programs", Naval Res. Log. Quarterly, Vol 28, No 3, pp 447-462.

135. *Spira P.M. and Pan A. [1975]*, "On Finding and Updating Spanning Trees and Shortest Paths", SIAM J. of Computing, Vol 4, No 3, pp 375-380.

136. *Syslo M., Deo N. and Kowalik J. [1983]*, "Discrete Optimization Algorithms With Pascal Programs", Prentice-Hall, New Jersey, U.S.A.

137. *Taha H.A. [1975]*, "Integer Programming Theory, Applications and Computations", Academic Press.

138. *Tarjan R.E. [1977]*, "Finding Optimum Branchings", Networks , Vol 7, pp 25-35.

139. *Turing A. [1936]*, "On the Computable numbers with application to the Entscheigsproblem", Proc. London Math.Soc. Ser. 2, Vol 42, pp 230-265.

140. *Vogel W.R. and Reinfeld N.V. [1958]*, Mathematical Programming,

Prentice-Hall, Englewood Cliffs, New Jersey, U.S.A.

141. *Wong R.T. [1986]*, "Location and Network Design",in Annoted Bibliography in Combinatorial Optimization , pp 129-147.