The capacitated arc routing problem: Lower bounds

Patrick Breslin BComm

Antony Keane BE

1997

ACKNOWLEDGEMENTS

We would like mostly to thank Mr. Peter Keenan, who was as much a partner as a supervisor, for his invaluable time and erudite guidance during the summer.

We must also pay tribute to our parents for their encouragement and support, not only for the duration of this dissertation, but throughout the year.

ABSTRACT

The Capacitated Arc Routing Problem is a *NP*-hard arc routing problem in which a fleet of vehicles, based on a specific vertex (the depot) and with a known capacity, must service the arcs of a graph on which demand exists, with minimal total cost and such that the capacity of the vehicles is not exceeded.

In 1992, three Japanese researchers devised the first exact algorithm to solve the Capacitated Arc Routing Problem optimally. In 1994, in an MMS dissertation, Montwill and Naughton implemented and improved this algorithm. Despite their success they found that the algorithm did not perform well when solving large problems, since it used too much computer memory and took too long.

The aim of our dissertation was to combat this problem. To this end our first objective was to implement recent developments in the field of lower bound heuristics for the Capacitated Arc Routing Problem. We concluded that lower bounds devised by three Spanish researchers were, as claimed, the tightest lower bounds to date. Our second objective was to modify and improve these lower bound heuristics. This second objective was indeed successfully achieved and our new lower bound heuristic returns the best results to date.

SUMMARY

Chapter 1 of our dissertation is an introductory chapter where we introduce the Chinese Postman Problem, on which the Capacitated Arc Routing Problem is based. We then explain the Capacitated Arc Routing Problem itself and outline the purpose of our dissertation.

Chapter 2 describes our main solution technique for solving the Capacitated Arc Routing Problem i.e. the Tour Construction algorithm as devised by Hirabayashi et al. and implemented and improved by Montwill and Naughton. With an understanding of the operation of the algorithm borne in mind, the motivation for our study of lower bound heuristics for the Capacitated Arc Routing Problem becomes apparent.

Chapter 3 clearly describes the most important of the existing lower bound heuristics. As presented in their original form these heuristics were written in a complex mathematical fashion with no attempt made to concisely explain their logic or reasoning. We have therefore undertaken to outline in a clear and comprehensible manner how these heuristics operate on graphs and thus return a lower bound figure.

Chapter 4 deals with our testing of the existing lower bound heuristics. We briefly describe a modification we made to the second of the Spanish bounds in order for them to cope with the time-capacitated problem. We include tables of results for both time-capacitated problems and volume-capacitated problems. We then analyse these sets of results by explaining the performance of the various bounds.

Chapter 5 describes the new lower bound heuristic which we devised. We clearly explain the foundation for our modifications and include tables of results displaying its performance against the best of the other lower bound heuristics. Our lower bound supersedes all known existing bounds.

Finally, we present our conclusions and possible areas of further research in *chapter 6*.

TABLE OF CONTENTS

CHAPTER 1	**Introduction**	**1**
CHAPTER 2	**Solution Technique for the Capacitated Arc Routing Problem**	**7**
	2.1 Matching Theory	
	2.2 The Node Duplication Transformation Procedure	
	2.3 The Tour Construction Algorithm	
	2.4 Practical Importance of Bounds to the T.C Algorithm.	
CHAPTER 3	**Existing Lower Bounds**	**25**
	3.1 Introduction	
	3.2 Bound LB1	
	3.3 Cutstes	
	3.4 Bound ZAW1	
	3.5 Bound LB2	
	3.6 Comments	
CHAPTER 4	**Testing the Existing Lower Bounds**	**48**
	4.1 Modification of LB2 for the Time-Capacitated Problem	
	4.2 Testing and Comparing the Bounds	
	4.3 Analysis of the Performance of the Bounds	
CHAPTER 5	**A Lower Bound for Sparse Networks**	**61**
	5.1 A New Successive Cutset Strategy	
	5.2 Analysis of L1	
	5.3 Lower Bounds and the Branch and Bound	
CHAPTER 6	**Conclusions and Further Research**	**72**
	6.1 Conclusions.	
	6.2 The Way Forward.	
REFERENCES		**77**
APPENDICES		**77**
APPENDIX A : Adjacency Lists		**77**
APPENDIX B : Program Listing		**77**

CHAPTER 1

INTRODUCTION

1.1 ROUTING PROBLEMS

Routing problems have been widely studied during recent years, mainly because of the increasing number of practical applications and the major increase in the costs associated with operating a fleet of vehicles. These problems can be divided into Node Routing Problems and Arc Routing problems. In Node Routing Problems the demand occurs in the nodes (or vertices) of a graph. An example of this might be a salesman who must visit each town in a country. Arc Routing Problems have demand on each of the arcs (or edges) of the graph. An example of an Arc Routing Problem might be a postman who must travel along each street in a town. Far more research has been done on Node Routing Problems, while Arc Routing Problems have received comparatively little attention despite their application to many problem areas, such as refuse collection, street sweeping operations, delivery of milk or post and inspection of distributed systems (electric power, telephone, or railway lines).

1.2 THE CHINESE POSTMAN PROBLEM

The Chinese Postman Problem (CPP) is a classification of Arc Routing Problems which involve finding a minimum cost cycle that traverses each arc in an undirected graph at least once. Consider a connected, undirected graph $G=(V, E)$ where V represents the set of vertices (or nodes) in G and E represents the set of edges (or arcs) in G. All arcs with a positive demand are called *demand arcs*. In practical terms, demand arcs represent streets which have letter boxes where letters are to be delivered. Each such arc will have three values associated with it:

1. ***Traversal Time*** (c_e) : Time to traverse arc (i.e. time taken to just travel down the street without carrying out any work on the street).

2. ***Service Time*** (c_e^*) : Time to service the demand arc (i.e. time taken to travel down the street and carry out some work on that street e.g. deliver letters while travelling down the street). Clearly $c_e^* \geq c_e$, as service time must include traversal time. Let C_T denote the sum of the servicing costs.

3. ***Volume*** (q_e) : The volume of work to be carried out on the arc (this would probably not constrain the postman problem as postal vans can carry an awful lot of letters, but would be very important in other Arc Routing applications such as rubbish collection).

We also specify a *depot node*. The CPP involves finding a minimum cost cycle that traverses each arc in E exactly once, starting and ending that cycle at the depot node. Figure 1.1 below shows a 9-arc graph which we will use as an example throughout this dissertation.

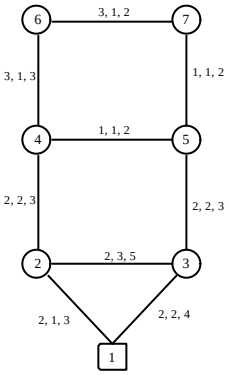


Figure 1.1 Weights on Arcs are listed in this order - **Volume, Traversal Time, Service Time**

If in a graph there exists a cycle that traverses each arc exactly once then the graph is *Eulerian*. The fundamental condition for an Eulerian graph is that every vertex be of even degree, i.e. every vertex has an even number of edges adjacent to it. This can be explained as follows.:

Suppose a graph is Eulerian, G, with an Euler Tour C. In traversing the tour C, every time C 'enters' a vertex by one edge, it 'leaves' by a different edge i.e. every time a vertex is encountered, two of the edges incident to that vertex are 'used'. Since all the edges incident with that vertex are in C, the vertex must

have even degree. Thus for a graph to be Eulerian, each vertex must be of even degree. A single tour which traverses all the arcs in such an Eulerian graph is called an *Euler Tour*. Fig. 1.2 shows an example of an Eulerian graph and a possible Euler tour.

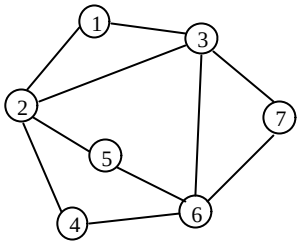


Figure 1.2 1-2-3-6-5-2-4-6-7-3-1 is an Euler Tour.

If the graph is not Eulerian, this means that some of the arcs will have to be traversed again, having already been serviced. A Minimum Cost Perfect Matching algorithm can be used to add artificial edges to match the nodes of odd degree. A feature of any graph is that the number of nodes of odd degree will be even (or zero). Once the artificial edges have been added, the graph will be Eulerian. Fig. 1.3 below shows an example of this process. A possible Euler Tour on this graph is (1-3-2-3-5-4-5-7-6-4-2-1).

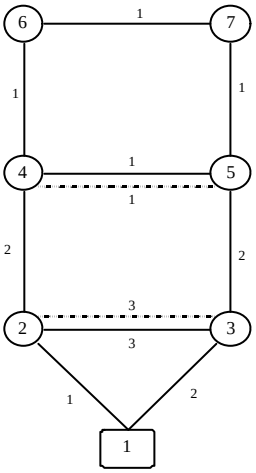


Figure 1.3 Euler Tour created using artificial arcs (2, 3) and (4, 5). The numbers shown are traversal times.

1.3 THE CAPACITATED ARC ROUTING PROBLEM

In this dissertation, we consider a special case of the CPP called the Capacitated Arc Routing Problem (CARP), in which a fleet of vehicles, with a known capacity (in terms of time or volume, or both), must service the arcs of a graph, with minimal total cost and in such a way that the capacity constraints are not violated. This adds another dimension to the problem in that more than one vehicle may be needed to service the graph. We define the CARP as follows:

For a given number of vehicles, M, each with a given volume capacity, Q_v, and/or a given time capacity Q_t :

MINIMISE $\sum$ Service times (c_e*) + $\sum$ Traversal times (c_e)

SUBJECT TO: (**1**) Each demand arc is assigned to one and one route only.

(**2**) Total volume for each route is $\leq Q_v$.

(**3**) Total time (service and traversal) for each route is $\leq Q_t$.

(**4**) Each route starts and finishes at the depot node.

Since the sum of the service times is constant for any given graph, the problem can be simplified in that the objective will be to minimise traversal times.

If, in our example, we specify that we have two postmen with a time capacity Q_t = 18 hours (Let Q_v equal infinity since size of postal van does not place a restriction on routes), then a feasible solution would be the two routes:

R_1 = (1~2~4-6-7-5-3-1) = 3 (Traversal) + 14 (Service) = 17 (Total Time)
R_2 = (1-2-4-5~3-2~1) = 3 (Traversal) + 13 (Service) = 16 (Total Time)

\- Serviced Arc ~ Traversed arc

These routes give a total time of 33 minutes of which 27 is service time the remainder is traversal time. An excellent review of Arc Routing Problems was presented by Eiselt, Gendreau, et al. [5], [6].

1.4 RECENT RESEARCH ON THE CARP

Golden and Wong[7] showed that the CARP is a *NP-hard* problem. Due to the complexity of the problem recent research has mainly been focused on developing and testing heuristics in order to obtain feasible solutions.

In 1992, Hirabayashi et al. [9] devised the first exact algorithm to solve the CARP optimally. In a 1994 MMS dissertation, Montwill & Naughton[12] implemented this algorithm and made a number of significant improvements.

In the area of lower bounds to the Capacitated Arc Routing problem, Golden and Wong[7] were the first to make a significant impact in 1981. They were followed in 1987 by Assad et al.[1]. However in 1988 Pearn [12] proposed a lower bound procedure which synthesised the above two methods and indeed was shown to dominate the Golden and Wong bound. Then in 1990 Hirabayashi et al [8] developed a lower bound procedure which they showed to be tighter than Pearn's bound.

1.5 THE PURPOSE OF THIS DISSERTATION

As successful as Montwill and Naughton were in implementing and improving the Tour Construction algorithm, they still encountered problems. Their main problem was as a direct result of the inefficiency and expansive nature of the Branch and Bound method. They found that for many problems they were prevented from finding optimal solutions due to the memory draining characteristic of the branch and bound algorithm, even with networks with as few as twenty arcs. For many problems the computer would run out of memory because of the sheer number of sub-problems generated by the algorithm.

By implementing new tighter bounds on the initial problem, the branch and bound method should reach an optimal solution in fewer sub-problems as few sub-problems are formed. This will increase the size of graphs which can be solved by the algorithm, and also have the added advantage of speeding up

solution times.

Benavent et al.[3] propose new lower bounds to the CARP which dominate other lower bounds to date as proposed by Golden and Wong and Win[13]. The principle aim of this dissertation is to implement these bounds and to make modifications to improve their performance on sparse networks.

CHAPTER 2

SOLUTION TECHNIQUE FOR THE CAPACITATED ARC ROUTING PROBLEM

In this chapter we review the solution technique implemented by Montwill & Naughton in their dissertation, as this technique is an integral part of our dissertation and our main solution finding tool. This review will lead to an understanding of how improved lower bounds can impact positively on this solution technique.

The technique, originally devised by Hirabayashi et al.[9] and implemented and improved by Montwill & Naughton is called the Tour Construction Algorithm. It is a branch and bound algorithm which uses matching and a graph transformation procedure called the Node Duplication Transformation which transforms a graph into a special type of graph suitably augmented for manipulation by the branch and bound algorithm.

2.1 MATCHING THEORY

In this section we discuss matching theory. Matching theory has a pivotal role to play in our dissertation as it is extensively used in both the Tour Construction algorithm and in our lower bound heuristics.

Let $G = (V, E)$ be a finite undirected graph. A *matching* in G is a subset of the arcs E, no two of which are incident with a common node. Given a matching M, a node which is not the terminal node of any arc in M is called an *exposed node*. A *perfect matching* M is a matching where there are no exposed nodes. Given real weights c_{ij} for each arc $(i, j) \in E$, to find the *minimum-cost perfect matching* (MCPM), we minimise $\sum (c_{ij} \mid (i, j) \in M)$ where M is a perfect matching. Fig 2.1 below shows a graph and it's MCPM.

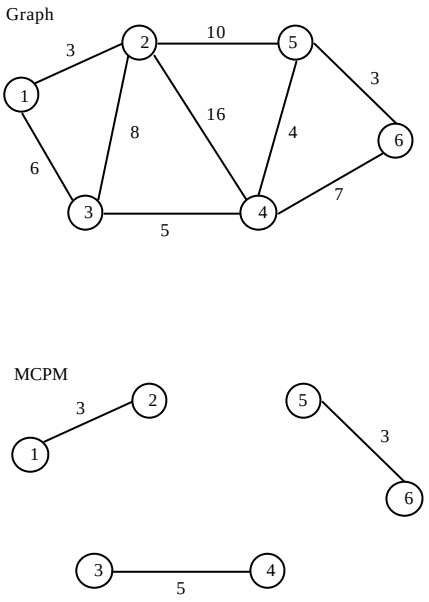


Figure 2.1 Minimum Cost Perfect Matching

The sum of the costs on the three arcs in the MCPM solution is 11 (i.e 3+3+5). No other perfect matching will give a lower total cost.

Not every graph will have a perfect matching. If there are an odd number of nodes or the arcs available do not allow for a perfect matching, then there will be exposed nodes. In our study, we are concerned with matchings on complete graphs with an even number of nodes. These conditions guarantee the existence of perfect matchings.

Montwill and Naughton[11] reviewed and tested three well known methods that have been developed in order to find a MCPM on a complete non-bipartite graph:

1. The Blossom algorithm (Edmonds [2]).

2. The Primal algorithm (Cunningham and Marsh [2]).

3. The Shortest Augmenting Path (SAP) method (Derigs [13]).

The minimum-cost perfect matching algorithm plays an important role in the Tour Construction algorithm since the MCPM is solved many times in order to determine the branching criterion at each sub-problem. It also is solved many times in the calculation of some of the lower bounds discussed in chapter 3, which reviews the existing lower bounds.

Algorithm(3) is a primal-dual method with cut inequalities and algorithm(2) is primal with blossom inequalities. Of these algorithms, the SAP is known to be the fastest. Montwill and Naughton used this algorithm for implementing the Tour Construction algorithm. This implementation of Derig's algorithm has been made available to us, so we also used Derig's algorithm in testing the lower bounds.

2.2 THE NODE DUPLICATION TRANSFORMATION PROCEDURE

Calculation of the Node Duplication Lower Bound (NDLB) involves the transformation of a given graph $G = (V,E)$ into a graph $G^t = (V^t, E^t)$ which is an augmented graph comprised of demand arcs and a number of artificial arcs. The significance of the artificial arcs will be explained later in this chapter.

(i) ***The Graph Structure***

We create the set of nodes, which represent copies of the nodes of the original graph, V, as follows :

For each node i $\in V$ which is adjacent to a demand arc, create the set called '*Family of i*' where Family[i] contains Degree[i] copies of node i from the original graph. Effectively for each node in V, we are making as many copies of the node as is equal to the degree of that node. Now we obtain the set V^D by forming the union of Family[i] for all i $\in V$.

Thus V^D = Family[1] $\cup$ Family[2] $\cup$ ... $\cup$ Family[n], where n is the number of nodes in the original graph.

We now create a set of nodes V^S, which represent copies of the depot, as follows:

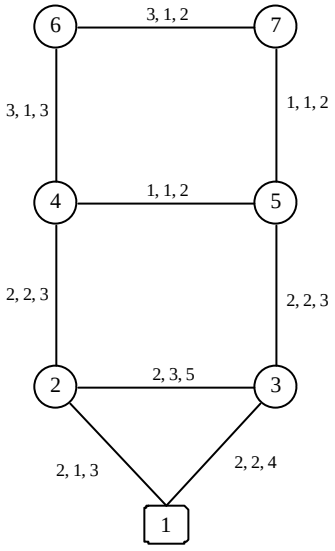
Letting 'M' denote the number of postmen who will service the network, the set V^S is comprised of (2 $\times$ M) nodes where each node represents a copy of the depot node, in the original graph G.

The set V^t is now obtained by forming the union of the two sets V^D and V^S.

Thus $V^t = V^S \cup V^D$.

(ii) ***The Demand Arcs***

We now have to assign each demand arc in G to some arc in G^t. For any demand arc (i,j) in G, we select a node in V^t from Family[i], which we denote by 'k', and another node in V^t from Family[j], which we denote by 'l'. Arc (k,l) is now set as the demand arc in G^t which corresponds to the demand arc (i,j) in the original graph G. The demand on this arc (k,l) is set equal to that of arc (i,j) in G. This is repeated for all demand arcs in such a way that no two demand arcs in G^t have common nodes. This is made possible by the fact that the number of nodes in Family[i] equals Degree[i] for any node $i \in V^D$.



Volume, traversal time, service time

Figure 2.2 The Original Graph

The transformation of our 9-arc 7-node graph to the transformed graph in Fig. 2.3 illustrates the transformation of the graph and the demand arcs.

The depot node is adjacent to two demand arcs, and thus Family[1] contains two copies of Node 1. We renumber these as nodes '1' and '2' in the transformed graph. We apply this procedure to each node in the original graph, while each time incrementing the corresponding node number in the transformed graph accordingly. If in this example we require M = 4 vehicles, we have 8 (2×M) copies of the depot node, referred to as 'depot nodes', numbered in this case as nodes 19 through to 26 inclusive. The demand arcs are then inserted into G^t, with arc (1,3) in G^t corresponding to arc (1,2) in G, arc (2,6) in G^t corresponding to arc (1,3) in G, and similarly for the remaining

demand arcs.

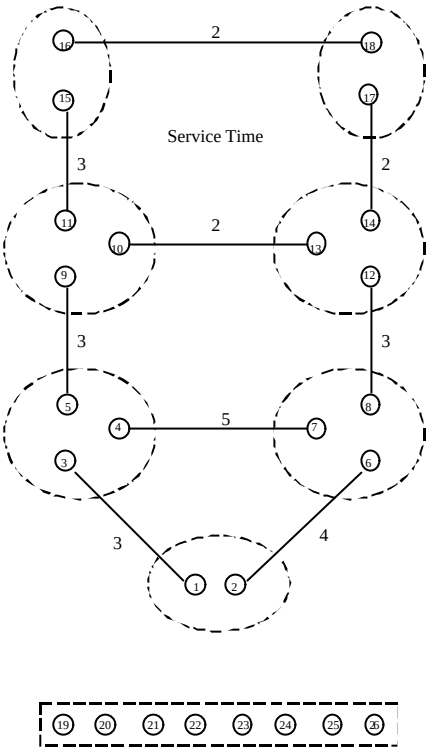


Figure 2.3 The Tranformed Graph

(iii) The Complete Graph

We now form a complete graph where the costs on the arcs are given as follows:

(a) The cost on each demand arc is set to infinity (i.e. they are prohibited from the MCPM solution)

(b) If two nodes are from separate families, the cost on the arc between them is set equal to the shortest path between their original nodes in the original graph G.

(c) If one node of an arc (i, j) is from a family (i.e. $i \in V^D$) and the other node is a depot node (i.e. $j \in V^S$), the cost on the arc between them is set equal to the shortest path from the node to the depot

(Node 1) in the original graph G.

(d) The cost on the arc between two nodes from the same family is set to zero.

(e) The cost on the arc between two depot nodes is set to infinity (i.e. these arcs are prohibited)

2.3 THE TOUR CONSTRUCTION ALGORITHM

The Tour Construction (T.C.) algorithm, as devised by Hirabayashi et al [9], is an algorithm based upon the Branch and Bound method, where an original problem is split into two sub-problems. Then each sub-problem is dealt with in turn and split into two further sub-problems, and so on. The crucial issues here are what determines the branching from each sub-problem into two further sub-problems, and what characteristic differentiates each of the two subsequent sub-problems.

The left sub-problem is characterised by an arc which is *prohibited* from the optimal solution. The right sub-problem and all child sub-problems will definitely *contain* that arc in the solution.

2.3.1 MCPM and the Tour Construction Algorithm

The T.C. algorithm uses a MCPM function which is passed a matrix of the costs on the arcs in the specific graph and returns the optimal matching and a lower bound figure. However, the algorithm may create a very large number of sub-problems before a feasible solution is found, thus requiring minimum-cost perfect matchings to be carried out many times.

The Lower Bound

The initial lower bound is now obtained by summing up the costs on the arcs in the initial optimal matching. This lower bound is a lower bound on the sum of the costs or traversal times for the optimal vehicle route given the matrix that is sent down to the MCPM function.

If we take our 9-arc example, the initial MCPM of the cost matrix is shown in Fig. 2.4 below. The costs are displayed on the arcs giving a lower bound of 39 (12 Traversal Time + 27 Service Time).

Finding a Feasible Solution

The significance of the Node Duplication Transformation now becomes clearer when the artificial arcs are added in to give the following graph :

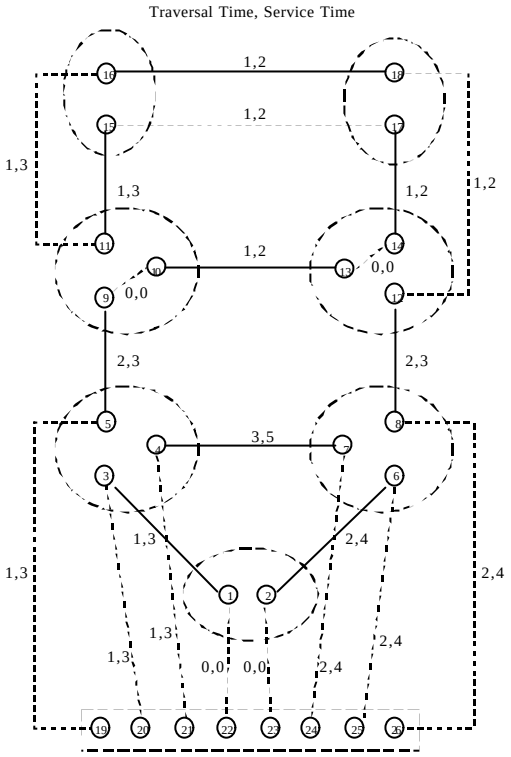


Figure 2.4 Initial MCPM on the 9-arc example.

The presence (or absence) of a feasible solution is based on the concept of an *alternating path.* Initially when an MCPM is performed on the node duplicated graph G^t and the demand arcs are subsequently added into the graph, we can observe the presence of alternating paths. These are paths whose arcs are alternately an MCPM solution arc(dotted lines) and a demand arc(solid lines). Thesewhich represent vehicle routes. An alternating path which starts and ends at the depot node and which does not violate either the time capacity constraint or the volume capacity constraint of the vehicle is called a *postman paths*. A collection of mutually disjoint postman paths where all demand arcs are contained in some postman path is called a *postman tour.*

In the example above, there are four alternating paths. Artificial arcs found by

the MCPM on the complete transformed graph are illustrated by dotted lines. When calculating the total time of the alternating paths, we use traversal times for MCPM solution arcs (or artificial arcs) and service time for demand arcs. The total times for the four alternating paths above are 4, 8, 6, and 21. This would be a postman tour if postman capacity was 21 or more. If capacity was less than 21, then the alternating path of length 21 cannot be a postman path, and a postman tour would not exist.

The Upper Bound

The algorithm sets the upper bound to the maximum lower bound possible. This maximum lower bound is the maximum total length of time possible to service the network, which is the product of the number of vehicles by the time capacity per vehicle. One unit of time is added to this and the total service time subtracted in order to give the upper bound.

2.3.2 THE TOUR CONSTRUCTION ALGORITHM

This algorithm is based on the branch and bound method. Each sub-problem is split into two further sub-problems according to a certain criterion. This criterion is the *branching arc* which is chosen from the MCPM solution. The left sub-problem has the characteristic that the branching arc can never be in the MCPM solution, while the right sub-problem has the characteristic that the branching arc must be in the MCPM solution.

For each arc (i,j) in the MCPM solution, the lower bound of the sub-problem with (i,j) prohibited from being in the MCPM solution is found. The branching arc (k,l) is the arc which returns the *highest* lower bound, when prohibited.

The Algorithm in Motion

Branching initially from the original problem, at each iteration work is carried

out on the right sub-problems, while saving the left sub-problems. Movement down the right hand side is continued until the sub-problem becomes infeasible or the sub-problem's lower bound is greater than or equal to the upper bound. The latter case is a possible occurrence as the the lower bound of each sub-problem is always less than or equal to the lower bounds of its left and right sub-problems.

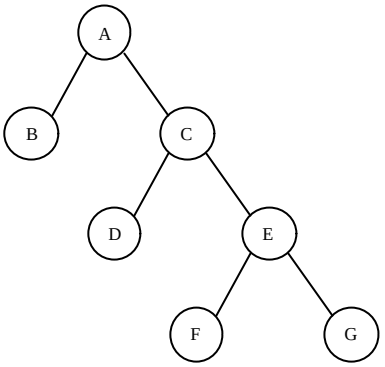


Figure 2.5 Branch and Bound Sub-Problems

The other occasion at which we stop at a sub-problem is when we have a postman tour. When this occurs the upper bound is set to the lower bound of the current sub-problem, and this sub-problem is stored as the *banker*. The banker is defined as being the best solution found so far. As the upper bound has been decreased, all stored left sub-problems whose lower bound is greater than or equal to the upper bound are deleted. (We ignore sub-problems whose lower bounds are equal to the upper bound because we are only looking for one optimal solution).

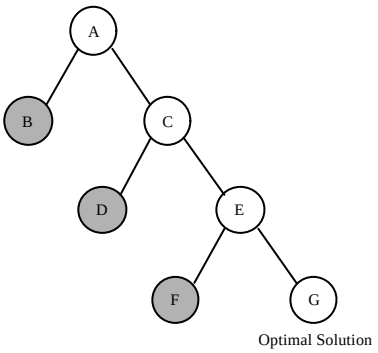


Figure 2.6 Fathomed Sub-Problems and Optimal Solution, Sub-Problem G

Now, if all remaining sub-problems have been deleted i.e. all branches have been fathomed, (as if Fig 2.6) our current feasible solution (banker) is thus our optimal solution (Sub-Problem G). Alternatively if a sub-problem existed with a lower bound less than the upper bound, then that sub-problem would be treated in the same way as sub-problem A.

Definite Arcs

A *definite arc* is a set of arcs which form a particular artificial arc. This artificial arc is a path of two or more arcs which are guaranteed to form part or all of a postman path in the final solution. Initially, at the beginning of the algorithm, the definite arcs are all the demand arcs.

To best understand the reason for the existence of definite arcs, it is necessary to examine the effect that the branching arc has. When we branch from a sub-problem to its right sub-problem, the branching arc is forced to be in the solution. This has the effect of making two definite arcs into one definite arc. For example (see Fig. 2.7 below), if arc (9, 12) is the first branching arc in the nine-arc example, we now have a definite arc from Node 5 to Node 8 as arcs (5, 9) and (8, 12) are demand arcs and arc (9, 12) must be in the solution. We also notice that the length of the definite arc has become 7 hours. An advantage of this is that Nodes 9 and 12 are excluded when finding sub-problem C's MCPM solutions (when finding C's branching arc) as they are automatically in the MCPM solution. Thus as we progress further down the right hand side, we have less arcs to be looked at when finding an MCPM solution. We also have less definite arcs but the definite arcs are getting longer. Therefore, at each right sub-problem we look at the new definite arc formed and see if we can prohibit more arcs from being in the MCPM solution. This shall be discussed further in section 2.3.4.

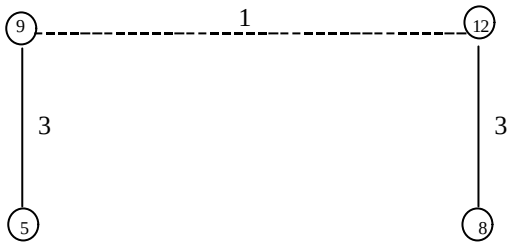


Figure 2.7 Definite Arcs

When we examine the transition from sub-problem C to sub-problem E, if the new definite arc causes an arc in C's (and thus A's) MCPM solution to be prohibited, it becomes necessary to find a new lower bound and MCPM for sub-problem E. Thus it is clear to see that until a new lower bound has to be found, the right sub-problems' lower bounds will remain the same, and hence at the same time less than the upper bound.

2.3.3 Outline Code of the Tour Construction Algorithm

Set U := ∅;

Transform network using Node Duplication;

Find Lower Bound and MCPM of the original problem;

Add original problem (Lower Bound and MCPM) to U;

Set Upper Bound := maximum possible lower bound + 1;

WHILE (U is NOT Empty) **DO**

Choose Sub-Problem with smallest Lower Bound from U.

WHILE (LB < UB) AND (sub-problem is feasible) **DO**

IF Postman Tour exists THEN

Upper Bound := Lower Bound;

Banker := Postman Tour;

Delete any Sub-Problems whose Lower Bound ≥ Upper Bound;

ELSE REPEAT

Select Branching arc *(x, y)*;

Add Sub-Problem to U where *arc (x, y)* is prohibited;

Make definite arc and do prohibit arcs;

UNTIL (Sub-Problem is feasible) OR (Arc in MCPM is prohibited);

IF *Arc* in MCPM was Prohibited THEN

Find new Lower Bound and MCPM.

END IF;

ENDIF;

ENDWHILE;

ENDWHILE;

Optimal Solution is in Banker;

ENDALG;

2.3.4 Prohibiting Arcs

There are several ways of prohibiting arcs throughout the Tour Construction algorithm in order to reduce the number of sub-problems generated.

(i) Initial Arc Prohibiting

Before we begin branching into sub-problems, we can prohibit arcs that can never be in the MCPM solution. This is carried out by going through each arc in the original transformed graph, not including the demand arcs. For each arc, the sum of the arc's length, the two demand arcs it connects, and the lengths of the shortest paths from the depot to the other end-points of the demand arcs, is found. If this sum exceeds the vehicle's time capacity, then the arc can never be in the MCPM solution. This initial arc prohibiting is of most use when the number of vehicles is large relative to the number of arcs. (Note that this prohibiting rule is altered for the volume-capacitated problem, i.e. it does not include any weight on traversed arcs, either the shortest paths back to the depot or the branching arc).

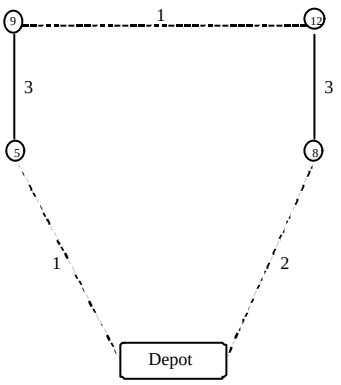


Figure 2.8 Initial Prohibiting

For example (see Fig. 2.8), if we look at arc (9, 12), it can never be in the MCPM solution if the postman capacity is less than 10 (ie. 1 + 3 + 1 + 3 + 2).

(ii) Prohibiting After Creating a Definite Arc

When a new definite arc is created, this definite arc can be used to prohibit

arcs in a similar approach to the initial arc prohibitng. The only arcs that need to be examined are the arcs incident to the end-points of the definite arc, as these are the only arcs which may have become infeasible.

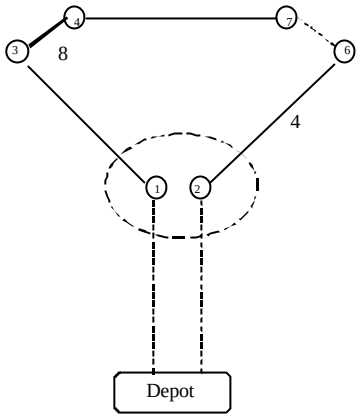


Figure 2.9 Prohibiting after creating definite arcs

For example, (see Fig 2.9), if arc (3, 4) is forced into the MCPM solution, a definite arc from Node 1 to Node 7 is formed. Therefore, arc (7, 6) can be prohibited if the postman's capacity is less than 12 (8 + 4).

(iii) Prohibiting Cycles

When a definite arc is formed, if an arc exists connecting the two end-points it must be prohibited as this will form a cycle which is not permissible in a postman path. For example, (see Fig. 2.10), if arc (9, 12) is in the MCPM solution, we must prohibit arc (5, 8).

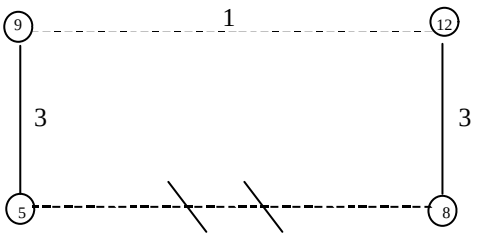


Figure 2.10 Prohibiting Cycles

(iv) Prohibiting Arcs with Depots

When saving a left sub-problem, the branching arc (k, l) is being prohibited

from ever being in the MCPM solution. If the branching arc includes a depot node (say that node ‘l’ is a depot node), the new MCPM solution of this left sub-problem will have the same lower bound with the non-depot node (node ‘k’) being connected to a different depot node in the new MCPM solution. In other words, arc (k, l) will be swapped with another arc (i, j) in the MCPM solution (say node ‘j’ is a depot node) creating two new MCPM arcs (k, j) and (i, l) with all the other MCPM arcs remaining the same. The branching arc of this new sub-problem would be the arc (k, j) and this situation will reoccur until ‘k’ has been paired with all the depot nodes.

The problem which arises here is that all these sub-problems are the same and thus only one of them really needs to be looked at. This problem is also magnified as each of these sub-problems will be broken down into the same set of sub-problems.

In order to combat this, at the stage when a left sub-problem is saved with the branching arc (k, l) containing a depot node, all arcs between the non-depot node and all other depot nodes are prohibited.

2.4 PRACTICAL IMPORTANCE OF BOUNDS TO THE TOUR CONSTRUCTION ALGORITHM

The concepts of lower and upper bounds for a particular problem together define the range of possible values within which the solution to that problem may lie. The wider the range, the greater the number of possible solutions.

In terms of the branch and bound algorithm, the most important practical issue to consider is how, as the algorithm proceeds, it drains the memory resources of the computer.

As previously outlined, the concept of the branch and bound algorithm is that from each sub-problem branches a left sub-problem and a right sub-problem. The left sub-problem is stored in the computer's memory, while work is carried out on the right sub-problem. A satisfactory and efficient programming implementation has already been developed so as to prevent storage of an entire cost-matrix at each sub-problem, involving the use of two trees of parent pointers [M & N]. Despite this minimisation of the amount of memory used at each sub-problem, in cases of larger networks, the sheer number of sub-problems generated eventually causes complete saturation of the computer's memory.

In order to address this issue and ultimately aim to reduce the number of sub-problems generated by the algorithm, an investigation into the bounds on the range of possible solution values becomes necessary. It is hoped that in this way, new improved bounds, both upper and lower, may be found in order to narrow the range of possible solution values. This in turn should reduce the number of sub-problems generated and also diminish the extent of profligate use of memory, thus allowing the algorithm to find the optimal solution within the computer's memory capacity.

In effect, it is desired to squeeze the range of the problem by *increasing* the lower bound as close to the optimal solution as possible, and by similarly *decreasing* the upper bound. This is what is meant by *tighter* bounds. By finding these tighter bounds at the beginning of the problem, it should inevitably lead to the generation of fewer sub-problems and hopefully allow

the algorithm to find the optimal solution (depending of course on the specific computer's memory resources). This is brought about by the fact that feasible solutions which may have been found in the wider solution range and which may have resulted from the generation of many sub-problems, may no longer be considered by the algorithm and not stored in memory. In this way a more direct attack, as it were, can be made on finding the optimal solution.

In this dissertation we focus on the *lower bound* with the aims of successfully implementing the tightest bounds to date for the Capacitated Arc Routing Problem and in turn making modifications to these bounds in order to improve their performance on sparse networks. Keenan and Naughton[10] have implemented upper bound heuristics. The heuristic procedure to find the lower bound for a particular network is effectively a pre-processing procedure, carried out on an augmented version of the network, which returns a bound as close to the optimal solution as possible, and indeed on occasion can return the optimal solution itself.

CHAPTER 3

EXISTING BOUNDS

This chapter reviews some of the lower bounds which have been put forward in recent years. Most of these bounds are presented in the literature in mathematical notation, and little effort is made to explain the logic or reasoning behind why bounds work well (or otherwise) or why one bound is better than another. The result of this is that it is difficult to understand the bounds and therefore difficult to modify the bounds to adapt them to specific problems or to specific types of graph. The bounds are each explained by stepping through our 9-arc example, and are accompanied by a mathematical summary. We use the same names and try to match the mathematical notation as much as possible to that of the article by Benavent, Campos, et al[3].

3.1 INTRODUCTION

The most basic lower bound to the CARP is the length of the Euler Tour attained by adding the artificial edges given by the MCPM on the nodes of odd degree to the graph (as discussed in Chapter 1). Each of the lower bounds discussed in this dissertation build upon this simple idea. Also, each of the bounds will require the resolution of a MCPM.

Unless special circumstances exist, many more traversals of arcs will occur in the optimal solution than those given by the matching of nodes of odd degree. This will be determined by the characteristic of the graph. A good lower bound is a bound which can make a good estimate of which arcs in the graph will have to be repeated, and how many times, without over estimating. Research has taken place to try to come up with sophisticated methods of estimating the required artificial arcs.

In this chapter we review the most recent lower bounding techniques which have been proposed. Each of these bounds represent different ways of analysing the characteristics of a graph in order to estimate the traversals. We trace the lower bounds as they have been developed, beginning with the more simplistic, leading on to the more recent complex methods.

In chapter 2, we reviewed the Tour Construction algorithm which gave an initial lower bound by doing a MCPM on the Node Duplicated graph. This was called the Node Duplication Lower Bound (NDLB). Our aim is to find a better lower bound than this.

Introduction to Lower Bounds

As they are presented in this chapter, the lower bounds have been interpreted from the published article by Benavent, Campos et al. [3]. Each of the bounds are discussed in terms of a graph, $G = (V, E)$, where V represents the vertices of the graph and E represents the edges of the graph, with which traversal

times and volume loads are associated. It should be noted that this problem is distinct from that discussed in chapter 2, where each edge had a traversal time and a service time only associated with it. These problems will be reconciled in chapter 4, where we explain the necessary adjustments in order to apply these bounds to the time-capacitated problem.

Some of the bounds discussed here involve an augmented graph, H_s, upon which a minimum cost perfect matching is carried out. This MCPM will be used to give an estimate of the extra traversals required to service G. The bounds differ on the way in which H_s is constructed.

H_s is a complete graph such that $H_s = (V_t, E_t)$, where V_t is made up of copies of nodes of G and some artificial vertices. The edges in E_t are the edges between the vertices in V_t. Some useful notation that will be used in the explanation of the bounds is:

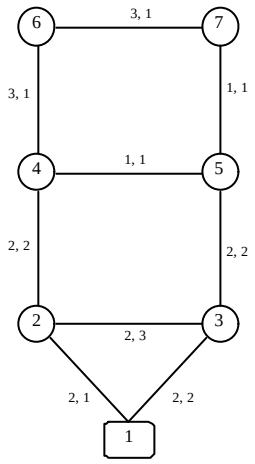
S : the sub set of nodes of odd degree in V.

C_T : the sum of the servicing times of the edges in E.

3.2 BOUND LB1

Thinking in everyday terms, where a post office is located at a village crossroads, it is quite rational to presume that the four roads meeting at that crossroads will be traversed many times. It is practical to think that a higher lower bound than the simple Euler Tour can be achieved by adding on an estimate of the extra traversals of those arcs adjacent to the depot. LB1 tries to capture this notion in more concrete terms.

Look again at our 9-arc example in Fig. 3.1. Suppose there are four vehicles, each with a capacity of 5. On each vehicle's route, it must travel 'out' of the depot along an adjacent arc, either arc (1,2) or arc (1,3), at the beginning of it's route. Similarly, at the end of the route, each vehicle must travel 'in' to the depot along one of these adjacent arcs. So in total, the arcs adjacent to the depot will be travelled along eight times, four times on outgoing journeys and four times on incoming journeys. But there are only 2 arcs adjacent to the depot. Thus we know that there will be at least six unnecessary traversals shared between the arcs adjacent to the depot.



Traversal Time, Volume

Figure 3.1 Our 9-arc example.

Include six artificial vertices in V_t which represent copies of the depot. Call this Set A.

Set A = $\{1_1, 1_2, 1_3, 1_4, 1_5, 1_6\}$.

These six copies of the depot are included in H_s so that six traversals of arcs adjacent to the depot will be included in the lower bound. For each of these six traversals, a copy of a vertex incident to the depot needs to be included in H_s. So we need to include enough copies of vertices 2 and 3 to pair off with the copies of the depot.

Choose the vertex closest to the depot. We choose the vertex closest to the depot to ensure that we do not over-estimate the traversal time. In our example, the shortest arc is arc (1,2), so we include a copy of Node 2 in V_t. When a vehicle meets Node 2, it must travel along (or have travelled from) another adjacent arc of that vertex to continue it's journey. But, Node 2 only has three adjacent arcs so it cannot match the six traversals needed for the depot. We include three copies of Node 2 in V_t. If we include any more copies of Node 2, then we force extra traversals of some arcs adjacent to Node 2. Again, since we do not want to over estimate the bound, we cannot include any more copies of Node 2. (This does not mean that H_s eliminates the possibility of more traversals of arc (1,2) as will be explained later). So, we look to the next closest vertex to the depot. In our example, this is Node 3. Node 3 also has three adjacent arcs. Between vertices 2 and 3, we include six artificial vertices in V_t. This is enough to offset the traversals 'to' and 'from' the depot.

Include three copies of Node 2 in V_t and three copies of Node 3. Call this Set B.

Set B = $\{2_1, 2_2, 2_3, 3_1, 3_2, 3_3\}$

Now, we need to represent the extra traversals needed due to the presence of nodes of odd degree in G. Of the remaining vertices in the graph, Nodes 4, 5, 6, and 7, two of them, Nodes 4 and 5, are of odd degree. These vertices must be matched with other vertices. So we need to create an artificial copy of each of these vertices and add them to V_t. Call this set C. Note that vertices of odd degree of which copies have been included in Set B cannot be included in Set C.

Set C = {4*, 5*}.

V_t = Set A Set B Set C.

Costs on the arcs in A_t are defined as follows:

- cost infinity between each of the copies of the depot, Set A, as vehicles cannot travel directly from one copy of the depot to another.
- between all other pairs of arcs, cost of the shortest path ascertained from the original graph will be used. Note that the cost between copies of the same node will be zero.

Carry out a MCPM on H_s. For our example, MCPM returns a value of 10. So LB1 provides a Lower Bound of 10 + Total Service Time = 10 + 14 = 24.

The optimal MCPM of the complete graph H_s returns the following matchings:

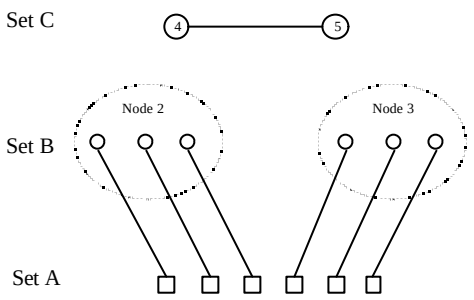


Figure 3.2 Optimal Matching of H_S.

In terms of our 9-arc example, LB1 has estimated that arc (1,2) will be traversed (or repeated) at least three times, as will arc (1,3). Also, arc (4,5) will be traversed at least once.

As mentioned earlier, including only three copies of Nodes 2 or 3 in Set B does not mean that the matching of H_s cannot represent more traversals of either of arc (1,2) or arc (1,3). This possibility is not restricted by the MCPM. Arc (1,3), and from Node 3 onwards to the next closest node of odd degree (Node 5) could be matched, but this will mean that some of the copies of Node 2 will be matched to another vertex, as in Fig. 3.3 below. **For our example,**

this is a non-optimal matching (i.e. it overestimates traversals and could give a bound greater then the optimal solution).

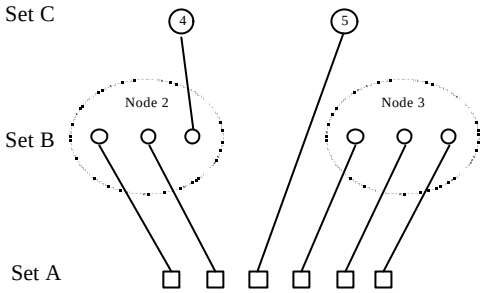


Figure 3.3 Matching where more copies of 2 are included.

Arc (1,5) which is shown in Fig 3.3 is representative of the shortest path from Node 1 to Node 5, calculated using the original distance in G from Node 1 to Node 5.

Summary of LB1

Assume that the vertices in G^r are 1, 2, ..., | V^r |, numbered in non-decreasing order with respect to their distances to the depot, so s_{12} s_{13} ... (where s_{1j} is the shortest path from the depot to Node j) and construct a complete graph G^r = (V^r, E^r) with V^r = A U B U C, where

Set A = {a1, ..., a_r} is a set of copies of the depot.

Set B contains Degree[i] copies of vertex i, for i = 2, ... , h where h is the minimum value of k such that Degree[2] + ... + Degree[k] >= r.

Set C contains a copy of each vertex in S (S contains the nodes of odd degree in the original graph, G) excluding the depot and those odd vertices whose copies are already included in Set B.

Cost on the edges of E^r are defined as follows: cost infinity between every pair of vertices in Set A, and, for all other pairs, the cost of the shortest path in G between the corresponding vertices of G^r. Note that the costs of the edges between copies of the same node will be zero except for the copies of the depot.

3.3 CUTSETS

Before we proceed, another important concept in Lower Bounds must be explained, that of cutsets. A formal definition of a cutset is as follows:

Let $G = (V, E)$ be an undirected graph. A *cut* (or cutset) in G is a set of arcs of G, described as follows:

X is a set of vertices of G such that $X \ V$.

$= V \setminus X$ and the cut

$(X,) = \{ | , \}$

Example (Fig 3.4 below):

If $X = \{1, 2, 3, 4\}$ and $= \{5, 6, 7, 8\}$

then the Cut(X,) = Cut(2,5, 3,6, 4,7)

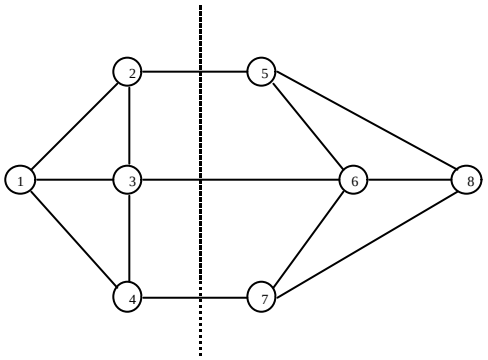


Figure 3.4 The cutset (2-5, 3-6, 4-7)

Looking at Fig. 3.4, we can see that the cutset splits the graph in two. (Note that this is a necessary, but not a sufficient condition to have a valid cutset). An easy way to develop a cut is to draw a line through a graph (such as the dotted line in Fig. 3.4 above) and any arc that crosses that line is in the cutset.

Later we will see how combining this concept of cutsets with that of matching make up the the fundamentals of some of the more recent Lower Bounds to the CARP. The difference in performance of the Lower Bounds depends on how these two properties of graphs are manipulated.

3.3.1 Cutsets and Lower Bounds

Consider LB1 again. We examined the number of times vehicles travelled along the arcs adjacent to the depot. The set of arcs adjacent to the depot would make up a valid cutset (as shown in Fig. 3.5 below). If these arcs were removed from the graph, the graph would be split into two components.

Component 1 = {1}

Component 2 = {2, 3, 4, 5, 6, 7}

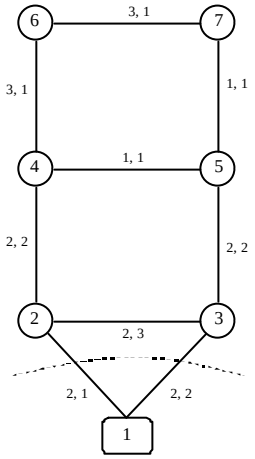


Figure 3.5 The Cutset from LB1.

When dealing with lower bounds, a cutset gives us a lot of information. Call arcs (1,2) and (1,3) the 'cutset' and the arcs (2,4), (2,3), (3,5), (4,5), (4,6), (5,7), (6,7) the 'cut sub-graph'. We will sometimes refer to the arcs in the cut sub-graph as arcs 'beyond the cut'. This means those arcs on the opposite side of the cut to the depot.

We will now outline some of the variables used in the rest of this chapter.

p = no. of vehicles to service the cut and the cut sub-graph.

q = the number of arcs in the cut.

r = 2p - q, the number of extra traversals required of those arcs in q.

c_s = the length of the shortest arcs in the cut.

For the example in Fig 3.5, these will be calculated as follows:

Assume 4 vehicles with capacity of 5.

At a vehicle capacity of 5 and with the given volume demands on the arcs, four vehicles are required to service the cut and the cut sub-graph.

p = 4

The number of arcs crossing the cut is 2.

q = 2

The number of extra traversals of arcs across the cut is 6. There will be four outgoing passings, and four incoming passings. Of these eight passings, only two can service arcs. So the other six passings must be traversals.

r = (2 4) - 2 = 6.

The shortest arc in the cut is arc (1,2). This has a length of 1.

c_s = 1.

3.4 BOUND ZAW 1

The first bound which we will discuss is from a Ph.D. carried out at the University of Augsburg, Germany by Zaw Win in 1987[13]. This bound is based on the concept of successive cutsets. Starting with Node 1 (call this set U), form a cut between U and (where is the set $V \setminus U$). In other words, form a cut between Node 1 and the nodes not in U (Nodes 2, 3, 4, 5, 6, 7) as in Fig. 3.5 above. The cut is Cut(1-2, 1-3).

Now calculate the number of vehicles required to service the cut and the cut sub-graph. In the graph above, four vehicles are required to service the cut and the cut sub-graph.

p = 4.

Therefore we know that there will be eight passings of arcs adjacent to the depot. Since there are only two demand arcs adjacent to the depot, we know that there will be at least six traversals between these arcs.

q = 2.

r = 2p - q = 8 - 2 = 6.

Choose the arc of least length from the arcs adjacent to the depot, arcs (1,2), (1,3) in our example. arc (1,2) is the shortest, with a traversal time of 1.

We multiply the number of vehicles required to service the cut and the cut sub-graph by the length of the shortest arc in the cut.

L1 = r c_s = 6 1 = 6.

Now we need to take the next cut. We take all nodes adjacent to to U, called U^* and include these in U.

U^* = {iV | i is adjacent to a vertex in U (not already in U)} and $U = U \ U^*$.

U^* = {2, 3}.

U = {1, 2, 3} and = {4, 5, 6, 7} in our example.

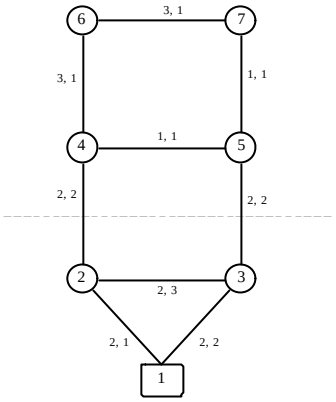


Figure 3.6 The second cut.

Our next cut will be arcs (2,4), (3,5) as in Fig 3.6. Looking at the service times on the arcs in the cut and cut sub-graph we can see that there are three vehicles required for service.

p = 3.

Again, there are two arcs in the cut.

q = 2.

r = 2p - q = 6 - 2 = 4.

Both arcs are of length 2. (But otherwise we would choose the shortest).

$c_s = 2$.

L1 = L1 + (c_s x r) = 6 + 8 = 14. This is our best bound to date.

The next cut contains arcs (4,6), (5,7).

p = 2, q = 2

r = 2p - q = 4 - 2 = 2

Both arcs have a traversing cost of 1, so $c_s = 1$.

L1 = L1 + (r c_s) = 14 + (2 1) = 16.

This is our best bound to date, and since there are no possible cuts left, this is

the Lower Bound given by ZAW1. ZAW1 can be summarised in algorithmic form as follows:

ALGORITHM ZAW1

Set $U = \{ 1 \}$, $L = L1 = L2 = 0$;

WHILE $U \ V$ **DO**

Let $V^* = V \setminus U$;

Let $G^* = (V^*, E^*)$ i.e. the graph induced having removed the node(s) in U from G.

The components of G^* are found. Suppose G^* has 'k' components $G^*_i = (V^*_i , E^*_i)$, for 1 i k.

$Cut(V^*_i) = \{i,j \ E \mid i \ V^*_i \ , j \ V \setminus V^*_i \}$

FOR i := 1 TO k DO

p_i := (q_e) / Q where q_e is summed over the edges E^*_i the edges in $Cut(V^*_i)$.

q_i := the number of edges in the set { e $Cut(V^*_i)$ | $q_e > 0$ }.

c_i := Min [c_e] e $Cut(V^*_i)$.

IF $r_i < 0$ THEN

IF q_i is even THEN Set $r_i = 0$

ELSE Set $r_i = 1$

ENDIF.

ENDFOR i.

$L1 = L1 + r_i \ c_i$ .... summed for each component i k.

Set U^* = the set of nodes {i V | i is adjacent to a vertex in U }.

$U := U \ U^*$.

ENDWHILE.

Set ZAW 1 = $L1 + C_T$.

ENDALG

3.4 BOUND LB 2

LB1 gives us a good bound because we capture the idea that the arcs adjacent to the depot will be traversed several times. Suppose now that we knew of other arcs in the graph that would be traversed. One such arc might be a bridge

connecting two towns in a postal district (or components on a graph), as illustrated in Fig. 3.7.

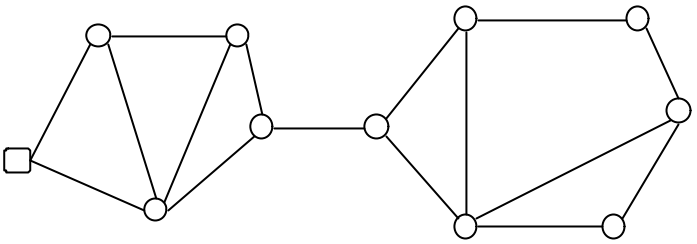


Figure 3.7 A bridge as a cut.

This is a very specific case however and will not arise often in real-life situations. However, the more general case may well arise where there is limited connectivity between components of a graph. An example of this might be how Dublin city is divided by the River Liffey. There are a limited number of bridges across the Liffey. Suppose that there was one post sorting office in Dublin, from which all vehicles began and finished their routes. Now suppose that sorting office is on the South side of the city. Now we also know that postmen will only work for 8 hours of the day, and in all, it takes 1000 hours to deliver the post for Dublin's North City. Therefor we know that at least 125 (1000 / 8) postal vehicles must cross the Liffey. Therefore, we know that between the 15 bridges that cross the Liffey in Dublin, these will be travelled across 250 times (125 2). This is the principle which LB2 tries to capture.

Bound LB2 further improves on Bound LB1 by considering *successive edge cutsets.* Consider again the example in 'Cutsets and Lower Bounds' above. Following a similar line of argument as was used for LB1 and using our 9-arc example, Set A, B and C will initially be as they were for LB1. One difference is that we will no longer refer to the elements of Set A as copies of the depot, for a reason which will become apparent later.

We need another set of Nodes for LB2, called Set D. Set D is not relevant for the cut at the depot, but it will be explained at the next cut, where it is more relevant.

Set A = $\{a_1, a_2, a_3, a_4, a_5, a_6\}$.

Set B = {2_1, 2_2, 2_3, 3_1, 3_2, 3_3}.

Set C = {4*, 5*}.

Set D = .

V_t = Set A Set B Set C Set D.

Costs on the arcs E_t is calculated as follows:

The cost on the edges between Set B Set C are the shortest paths between the corresponding vertices in the original graph. Note that the cost on edges between copies of the same vertex will be zero. For edges between i Set B Set C and j Set A Set D, the cost on the edge is the minimum distance from Node i to any node in U.

The MCPM on H_s returns a value of 10 (the same value as LB1).

Now, we have found a lower bound using the information from the Cut(1-2, 1-3) and the matching of the remaining nodes of odd degree (Nodes 4 and 5). But, further research by Zaw Win (further developed by Benavent et al) has shown that more information can be gained by looking at successive edge cutsets, i.e by moving away from the depot out into the graph in search of 'good cuts' such as the River Liffey.

The current lower bound is 10 + total service cost = (10 + 14) = 24.

Since the cost of servicing the graph is constant, let us assume from now on that when we refer to the lower bound, we refer to that time that is added on to the servicing time. In our case 10 would be the lower bound.

Remember that the shortest arc out of the depot is arc (1,2), with a traversal time of 1. Remember also that six additional traversal of arcs out of Node 1 had to be made. Store a value, L1 (which equals the value for L1 in ZAW1 at the same cut).

L1 = cs r.

c_s = 1

r = 6

L1 = 6.

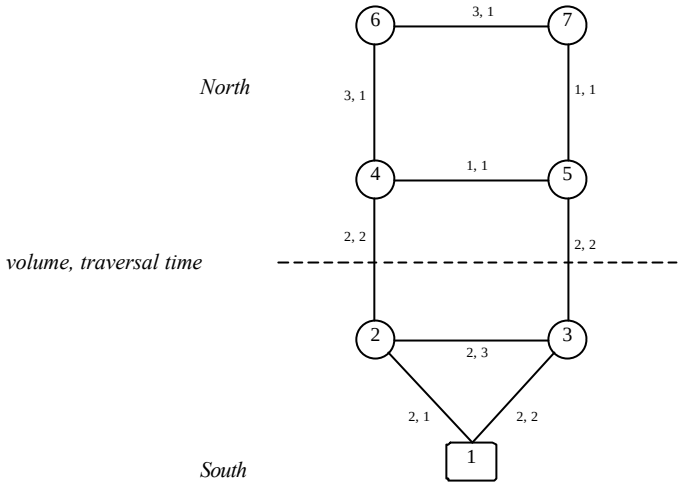


Figure 3.8 The second cut.

Now suppose that a river cut across the graph as shown by the dotted line in Fig. 3.8 above. This forms a natural cut, Cut(2-4, 3-5).

$U = \{1, 2, 3\}$.

Suppose that this was the Liffey dividing Dublin's North and South sides. The North Side of the graph has a demand of 12 (including the bridges across the cut). With a vehicle capacity of 4, three such vehicles are required to cross the river to service the North side of the graph. We also know that each of these three vehicles will have to return. So, in total, the river will be crossed six times, three times on outgoing traversals and three on incoming traversals. However, there are only two demand arcs that cross the cut. So, four out of the six times that the Liffey is crossed, a bridge will be traversed without servicing. Create four artificial vertices which can represent nodes on the South side of the graph.

Set $A = \{a_1, a_2, a_3, a_4\}$.

Now, in order to provide for the four artificial edges which will create the traversals across the cut, we need to match Set A with copies of the nodes on

the North side of the cut. Following a similar argument as was used to create Set B in LB1, 3 copies of Node 4 will be created and 3 copies of Node 5 will created since both nodes have degree 3.

Set B = $\{4_1, 4_2, 4_3, 5_1, 5_2, 5_3\}$.

Also, within the North Side, there may be nodes of odd degree (not already included in Set B) which need to be matched with each other. Looking at these nodes, Nodes 6 and 7, we can see that there are no such nodes in this instance.

Set C = .

Including Set C in V_t represents the odd nodes not yet included in V_t. Odd nodes between the depot and the cut are not included. Including only some of the odd nodes from theoriginal graph might possibly result in a sub-optimal matching (i.e. a matching which is too high). This might occur if a MCPM of odd nodes in the original graph involved matchings across the cut. In other words, if a node of odd degree beyond the cut would be matched with a node of odd degree on the depot side of the cut in a MCPM on the odd nodes in the original graph, this cannot be catered for by a matching on H_s.

L1 = c_s r = 4 2 = 6 + 8 = 14.

So we include artificial vertices in V_t which will prevent this occuring. The number of artificial nodes created for Set D is equal to the number of nodes in Set C minus r. If this number is negative, Set D is empty.

Set D = max{0, |Set C| - r}.

Set D = .

Vt = Set A Set B Set C Set D.

Set the costs on A_t, the arcs in the complete graph H_s using the same rules as before.

The MCPM on this graph returns a value of 8. At first sight, this bound is

inferior to the initial bound of 10. However, if we consider the L1 value of 6 calculated earlier, which is the estimated traversals behind the cut and add this to the new bound of 8, we get a total of 14. This gives a better lower bound than the 10 given earlier, so we save this as our new lower bound.

Current Lower Bound = 14.

Create the next cutset.

U = {1, 2, 3, 4, 5}

Using the same principles as before:

Set A = {a_1, a_2, a_3, a_4} as two vehicles needed to sevice arcs (4,6), (5,7), (6,7).

Set B = {6_1, 6_2, 7_1, 7_2}

Set C =

Set D =

V_t = Set A Set B Set C Set D.

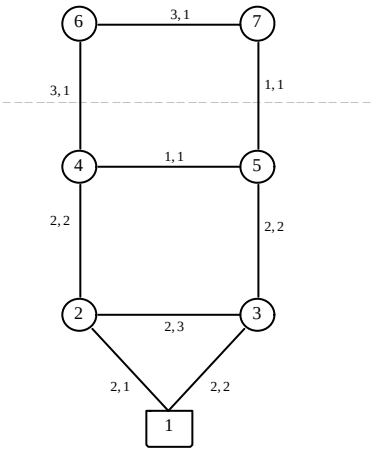


Figure. 3.9 The Final CutSet.

Again, form the complete graph H_s=(V_t, E_t) and carry out a MCPM. The MCPM on this graph returns a value of 2. Costs on the edges of E_t are calculated as before. Adding the estimated traversals behind the cut, 14, to 2,

we get a new lower bound of 16. Since this is higher than any bound seen before we store this as our bound.

Current Lower Bound = 16.

Now take all the adjacent nodes of the cutset. We see that all nodes in E are now a member of U, and so the Lower Bound Heuristic stops.

LB2 = 16. Adding this to the total service time for the graph, we get a lower bound of 30. The optimal solution to this graph is 32. So 30 appears to be a good lower bound.

LB2 can be summarised in algorithmic form as follows:

ALGORITHM LB2

Set $U = \{ 1 \}$, $L = L1 = L2 = 0$;

WHILE $U \neq V$ **DO**

Let $V^* = V \setminus U$;

Let G^* be the graph (V^*, E^*) i.e the graph induced having removed the node(s) in U from G.

The components of G^* are found. Suppose G^* has 'k' components $G^*_i = (V^*_i, E^*_i)$, for 1 $\leq$ i $\leq$ k.

$Cut(V^*_i) = \{e = (i,j) \in E \mid i \in V^*_i, j \in V \setminus V^*_i \}$

FOR i := 1 TO k DO

p_i := (q_e) / Q where q_e is summed over the edges E^*_i the edges in $Cut(V^*_i)$.

q_i := the number of edges in the set $\{ e \in Cut(V^*_i) \mid q_e > 0 \}$.

c_i := Min [c_e] $e \in Cut(V^*_i)$.

r_i := Max [0, ((2 p_i) - q_i)]

Let $S^*_i = S \cap V^*_i$.

IF S^*_i OR $r_i > 0$ THEN

Construct a weighted graph H_S .

(see below for construction of H_S) .

Set m_i = MCPM (H_S).

ENDFOR i.

$L := \sum m_i$ ..... summed for each component i $\leq$ k.

$L2$:= Max [$L2$, $L + L1 + C_T$].

$L1 := L1 + \sum r_i \; c_i$ .... summed for each component i $\leq$ k.

Set U^* = the set of nodes $\{i \in V \mid$ i is adjacent to a vertex in $U \}$.

$U := U \cup U^*$.

ENDWHILE.

Set LB 2 = $L2$.

ENDALG

Construction of Complete Weighted Graph H_S for LB 2

Preliminary

Let *Mdist*(i) denote the minimum distance between vertex 'i' in V^*_i and any vertex of U. Suppose that the vertices in V^*_i are renumbered i_1, i_2, ... in such a way that *Mdist*(i_1) *Mdist*(i_2) .....

Let h be the minimum integer such that *Degree*(i_1) + ...+ *Degree*(i_h) r_i.

The Vertex Set

Graph H_S is a complete weighted graph whose set of vertices is comprised of four sets :

SetA SetB SetC SetD

Set A is a set of artificial vertices, where = r_i

Set B contains *Degree*(i_k) copies of vertex i_k V^*_i for k = 1....h.

Set C contains a copy of each vertex in S^*_i except for those copies who are already in SetB.

Set D is a set of artificial vertices, where = Max [0, (number of nodes in S^*_i) - r_i]

The Arc-Cost Matrix

Let w_{ij} represent the cost on edge (i,j) in H_S.

The costs on the edges in H_S are infinite except for the following:

- if i, j {Set B Set C} then w_{ij} = the shortest path between the corresponding vertices in *G*. (Note that w_{ij} = 0 if i and j are copies of the same vertex).
- if i {Set B Set C} and j {Set A Set D} then w_{ij} = *Mdist*(k_i) where k_i denotes the corresponding vertex to i in *G*.
- if i, j Set D then w_{ij} = 0.

3.5 COMMENTS

In 1987 Win produced the ZAW1 bound which estimated the number of traversals across a cut. This bound was rather simplistic, and as we will see in chapter 4, rarely gives a tight bound. In the same Ph.D. he describes a bound, ZAW2, which is based on the same cutsets as ZAW1, but involves a matching on an augmented graph which takes advantage of some of the characteristics of the graph. We did not discuss this bound in this chapter since Benavent et al. prove in their paper that LB2 supersedes ZAW2.

Benavent et al. adapted LB1 to be applied to the successive cutsets in a very similar algorithm to ZAW2, but sometimes gives improved bounds. This is the bound LB2 as described in the last section. This is confirmed in the analysis of the next chapter. Bound LB2, to the best of our knowledge is the best bound which has been published to date.

In their article, Benavent et al. deal with bound LB1 and LB2 seperately, and described them as different bounds. When programming these bounds, we found that the first iteration, or the first cut at the depot as it were, gives the same bound as LB1. Though described using different notation, bound LB1 takes a cut at the depot.

CHAPTER 4

ANALYSIS OF THE EXISTING LOWER BOUNDS

In this chapter we test the performance of the various lower bound heuristics.

By way of doing this we test the two main bounds devised by Benavent et al. [3] against the bounds of Zaw Win [13] and Hirabayashi et al. [8] on a series of time-capacitated networks and volume-capacitated networks.

We confirm the claim by Benavent et al. that the second of their bounds, LB2, outperforms the bounds of Zaw Win. We also find that LB2 is at least as good as the bound of Hirabayashi et al. and indeed is often better.

4.1 MODIFICATION OF LB2 FOR THE TIME CAPACITATED PROBLEM

For the Capacitated Chinese Postman Problem as implemented by Montwill and Naughton, time was the only constraint on each vehicle, or route. However, the bounds as written by Benavent et al. And Win deal with the case where there is a capacity in terms of volume or load, i.e. how much each vehicle can physically carry. Corresponding to this volume capacity on each vehicle, there was also a required volume on each arc i.e. how much each vehicle must deliver on each arc. We needed to modify the lower bound in order to accommodate problems where the vehicles' capacities are given in terms of time.

At each cut, the number of vehicles required to service the cut and the cut sub-graph is calculated given a fixed vehicle capacity. When the vehicle capacity is in terms of time, then the shortest path from the cut to the depot must be subtracted from this capacity. Using this new reduced vehicle capacity, we calculate how many vehicles are required to service the graph.

We explain this as follows:

For each arc in the cut, the terminal node of the arc that is closest to the depot is an element of $\delta(V)$.

Let $\delta(V) = U \cap \text{Cut}(V^*)$

sp_i = Shortest Path from Node i, where $i \in \delta(V)$, to the depot

$Q_{mod} = Q - \text{Min}[sp_i]$.

Q_{mod} is the new vehicle capacity adjusted for the time capacitated problem.

Using our 9-arc example, this can be explained as follows:

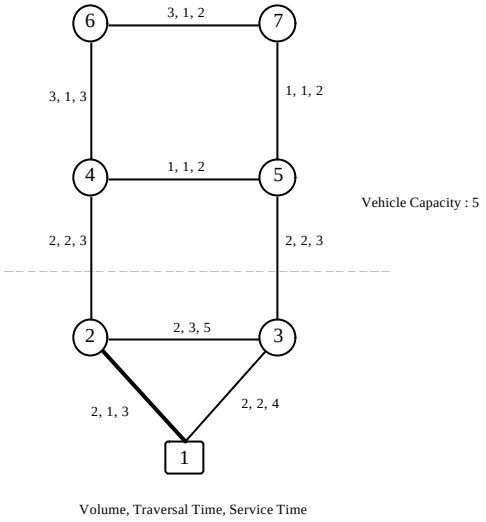


Figure 4.1 The 9 arc example

Suppose vehicle capacity is 5 hours for the graph in Fig. 4.1. At the second cut, as in LB2 above, 4 vehicles are required to service the cut and the cut sub-graph. Between the arcs that lie on the cut, and the arcs which are in the component beyond the cut it takes 15 hours to service arcs (2,4), (3,5), (4,5), (4,6), (4,7), (6,7). Since vehicle capacity is 5 hours, it will take at least three vehicles to service that section of the graph. But, each time a vehicle crosses the cut, it will be travelling to or from the depot, which will take at least one hour of traversing time (along the highlighted arc, which is the Shortest Path from the cut to the depot). So, in fact, once a vehicle has reached the cut, it now only has a capacity of 4 hours, since it has spent one hour travelling from the depot. So, given that it will take fifteen hours of service, it will take at least four vehicles of capacity 4 to service the cut and beyond.

The lower bounds presented in this chapter have been implemented on a series of graphs which have been extracted from the rural Irish road network (in the case of the time-capacitated problem) and on graphs which have been obtained from Hirabayashi et al. (in the case of the volume-capacitated problem). In the following tables of results, Q_v represents the vehicle capacity, while V represents the number of vehicles used.

4.2.1 Time-Capacitated Problems

The time-capacitated problem exists when the demand on the arcs is stated in terms of some time (or distance). The time factor represents the time it takes to service an arc or to traverse an arc. When calculating the total time of a route, both the service times and traversal times must be included. Total service time plus total traversal time must not be greater than vehicle capacity.

The adjacency lists of these graphs are attached in appendix A.

10 Arc Problem

Q_v	V	LB 2	LB 1	ZAW 1	ZAW 2	NDLB
125	3	61	51	53	61	51
150	3	51	51	43	51	51
175	2	37	37	29	37	37
200	2	37	37	29	37	37

16 Arc Problem

Q_v	V	LB 2	LB 1	ZAW 1	ZAW 2	NDLB
125	4	87	83	64	87	83
140	4	83	83	56	83	83
200	3	65	65	38	65	65
250	2	47	47	20	47	47

25 Arc Problem

Q_v	V	LB 2	LB 1	ZAW 1	ZAW 2	NDLB
135	5	143	120	115	143	108
175	4	107	92	79	107	92
200	4	95	92	67	95	92
250	3	79	70	51	79	70

34 Arc Problem

Q_v	V	LB 2	LB 1	ZAW 1	ZAW 2	NDLB
120	7	175	148	155	175	154
170	5	113	108	91	113	108
200	4	97	97	75	97	90
250	4	90	90	63	90	90

45 Arc Problem

Q_v	V	LB 2	LB 1	ZAW 1	ZAW 2	NDLB
150	7	221	194	161	219	194
200	5	163	146	105	163	146
250	4	131	126	73	131	126
300	3	106	106	41	106	106

50 Arc Problem

Q_v	V	LB 2	LB 1	ZAW 1	ZAW 2	NDLB
150	7	197	185	146	192	185
175	6	169	161	122	168	161
200	5	141	137	90	141	137
250	4	117	117	58	117	117

60 Arc Problem

Q_v	V	LB 2	LB 1	ZAW 1	ZAW 2	NDLB
150	7	178	135	128	178	165
175	6	152	121	100	152	121
200	5	130	108	76	128	108
250	4	112	98	52	112	98

4.2.2 Volume Capacitated Problems

The volume capacitated-problem exists when the demand on the arcs is stated in terms of some volume or load. Volume or load represents some physical amount which is attributed to each arc. Vehicle capacity will be stated in terms of how much a vehicle can physically carry.

The results presented below have been taken from a selection of data files obtained from Hirabayashi at al. Each network possessed fifteen arcs, and the number of nodes in each ranged from six to ten. Vehicle Capacity, Q_v, is 150.

The adjacency lists of these graphs are also attached in Appendix A.

15 Arc / 7 Node Problems

Graph	LB 2	LB 1	ZAW 1	ZAW 2	NDLB
aa10	1238	1238	1206	1238	1238
aa11	2542	2542	1947	2542	2542
aa13	2605	2605	2484	2605	2605
aa14	3299	3299	2162	3299	3299

5 Arc / 8 Node Problems

Graph	LB 2	LB 1	ZAW 1	ZAW 2	NDLB
aa01	1617	1617	1384	1617	1617
aa03	1697	1697	1210	1697	1697
aa05	1019	1019	430	1019	1019
aa18	1679	1679	946	1679	1679

15 Arc / 9 Node Problems

Graph	LB 2	LB 1	ZAW 1	ZAW 2	NDLB
aa19	1672	1672	792	1672	1672
aa20	1759	1759	1272	1759	1759
aa21	1520	1520	447	1520	1520

15 Arc / 10 Node Problems

Graph	LB 2	LB 1	ZAW 1	ZAW 2	NDLB
aa07	5361	5361	4287	4773	5361
aa08	2547	2547	1610	2547	2547

4. 3 ANALYSIS OF THE PERFORMANCE OF THE BOUNDS

In the analysis of volume capacitated networks, we found that on very few occasions was there any difference between LB1, ZAW2 and LB2 on the volume capacitated problems. It should be borne in mind that this can be attributed to two factors.

(1) The Japanese data files were 15-arc problems. This is a relatively small sized graph and therefore we should not expect large differences between the bounds.

(2) All of these graphs are well connected and are not sparse in nature. Therefore, it is unlikely that a 'good cut' will exist apart from the cut at the depot, and thus LB2 and ZAW2 are unlikely to have any great improvement over LB1.

With regard to the volume-capacitated problems, on the files made available to us by the Japanese researchers, the NDLB was always matched but never outperformed. Furthermore LB1 matched LB2 on each graph, and LB2 only outperformed ZAW2 on one occasion. The poor performance of these cut-based methods against NDLB can be explained when we look more closely at the nature of the volume-capacitated graphs, and in particular at their connectivity.

For these reasons, the rest of this analysis will concentrate on the time-capacitated graphs taken from the rural Irish road network.

Our results support the claim by Benavent et al. that bound LB2 is no worse than bound LB1 and the bounds of Zaw Win. ZAW1 is the worst bound. Only on one occasion did it beat any of the other bounds. It beat LB1 on a 10-arc time capacitated problem. ZAW2 proved to out-perform LB1 on a high number of occassions. Only on a very few number of these occassions did LB2 make a further improvement over ZAW2.

The following table shows how each of the bounds compared to ZAW1. We have used ZAW1 as the base data as this is the worst of the bounds and we

measure performance as improvement over ZAW1. The figures shown are an average improvement from the data in the tables taken from time-capacitated problems.

BOUND	% IMPROVEMENT
LB2	46.06 %
LB1	33.26 %
ZAW2	45.60 %
NDLB	31.47 %

Table 4.1 Average Percentage Improvement of Bounds over ZAW1

LB2 was never beaten by NDLB. The only case when NDLB beat LB2 is if the number of vehicles is more than is required to service the graph at a given vehicle capacity. The reason that NDLB will beat LB2 on these occasions is that NDLB is based on the number of vehicles, whereas LB2 is based on the vehicle capacity (it calculates the number of vehicles required to service the graph by looking at vehicle capacity and the weight on the arcs in the graph). We did not test these cases, as a minor adjustment to LB2 will amend this difference. This adjustment would force the number of vehicles crossing the first cut in LB2 to be equal to the number of vehicles specified.

In keeping with the findings of Benavent et al [3], LB2 outperforms ZAW2 when the vehicle capacity is low relative to the total demand i.e. there are more vehicles and each vehicle will service fewer edges. LB2 is at least as good as NDLB, on an appreciable number of occasions LB2 is indeed a tighter bound than NDLB.

4.3.1 Graph Connectivity

If we look at the workings of LB2 in more detail, it is clear that as successive cuts are carried out, the cut sub-graph becomes smaller i.e. it contains fewer nodes and arcs. Therefore it is generally the case that the MCPM on H_s , the augmented complete graph generated from the particular cut you are looking at, also becomes smaller. In other words, the MCPM on a matrix H_s generated as a result of a cut beyond the depot (i.e. not at the depot) will rarely be greater than the MCPM returned as a result of the cut at the depot. Thus, recalling that bound LB1 is based purely on the cut at the depot, and that bound LB2 carries out successive cuts following the cut at the depot, it becomes clear that in order for LB2 to improve upon LB1, there is a strong reliance on *L1*, the estimate of traversals between the depot and the cut. That is, in order for LB2 to return a better bound than LB1, one of the cuts away from the depot combined with *L1* must give a higher value than the cut at the depot, i.e. $MCPM(H_S) + L1 > L2$, where *L2* is the highest bound found so far. With this reliance of LB2 on the value calculated for *L1*, a closer examination of occasions when graphs might return a high *L1* value was necessary.

Recall that $L1 = L1 + \sum (r \times c)$

where $r = \text{Max}[\,0, (2 \times p) - q\,]$

i. e. the additional traversals required across the cut

and p = no vehicles required to service the cut and cut sub-graph,

and q = number of arcs in the cut.

and c_s = length of the shortest arc in the cut.

For any given cut, the number of vehicles having to cross the cut is *p*. So, as the number of arcs on the cut decreases, the value for *r* increases and conversely, as the number of arcs in the cut increases, the value for *r* decreases.

Considering the cut at the depot, if the depot is well connected (i.e. if it is of

high degree) this cut will result in an *L1* value which is too small to cause *L2* (the best bound to date in LB2) to change.

This explains the poor performance of LB2 on the Japanese data files that were used in testing. The Japanese files are very well connected, and the values for *L1* were never large enough to impact on the bound, and for each graph, the tightest bound obtained came from the initial cut at the depot. This clarifies why LB2 was never better than LB1 for the volume-capacitated Japanese files.

4.3.2 Vehicle Capacity

If vehicle capacity is low relative to the demand on the graph, then LB2 will also perform better. This is in line with the findings of the Benavent at al.[3]. The reason that this occurs is that as the vehicle capacity decreases, the number of vehicles required to cross the successive cuts becomes larger. This means that there will have to be more traversals across the cutsets resulting in a higher lower bound.

To summarise, LB2 is found to be the best lower bound, and this can be attributed to the fact that it sythesises the best features from the other lower bounds.

CHAPTER 5

A LOWER BOUND FOR SPARSE NETWORKS

Bound LB2 has clearly performed the best during the testing phase of this dissertation. In this chapter we describe the modifications which we made to the LB2 to try to take advantage of the sparse nature of the graphs which we were working with. This involves using a method to look at more of the cuts than LB2. We will also analyse where we feel significant improvements can be made to the bound. We also analyse how these improvements will affect the branch and bound.

5.1 A NEW SUCCESSIVE CUTSET STRATEGY

At the general stage, LB2 takes all nodes adjacent to nodes in the existing U, a set which we call U^*, and adds those to the U set. The resulting cut is formed by including all arcs from the existing graph which connect the new U set and the component, V^*, formed by removing the nodes in U from the original graph.

Stated formally :

Set $U^* = \{i \in V \mid i \text{ is adjacent to a vertex in } U\}$.

$U = U \cup U^*$.

Set $V^* = V - U$.

At the general stage, the cut is formed

$Cut(i, j) = \{ij \in E \mid i \in U, j \in V^*\}$.

Since LB2 increases the U set by the number of nodes in U^* on each iteration, it only examines a small number of cuts relative to the total number of possible cuts in the graph. We have implemented a method which increases the size of U by one on each iteration by doing an iteration of the bound adding in the nodes in U^* *one at a time.*

When adding nodes one at a time, we devised a selection procedure to decide in what order to add the nodes from U^*. Except in exceptional circumstances, there will be more than one node in U^*. The procedure that we came up with selects nodes in increasing order of degree. In other words, we looked at each of the nodes in U^* and added the node with the smallest degree to U first. We marked that node so that it will not be considered again. We carried out an iteration of LB2 as before. Then look for the node in U^* with the next lowest degree (which is not marked) and include that node in U. Repeat this procedure until all nodes from U^* are included in U. Once all the nodes from U^* are included in U, we need to get a new U^*. This is attained in the same manner as before. U^* is the set of nodes adjacent to nodes in U. This

procedure will include nodes in *U* in a Breadth First Search type pattern.

(We do not use our 9-arc example here, as the degree of the nodes in each successive set *U** are all the same)

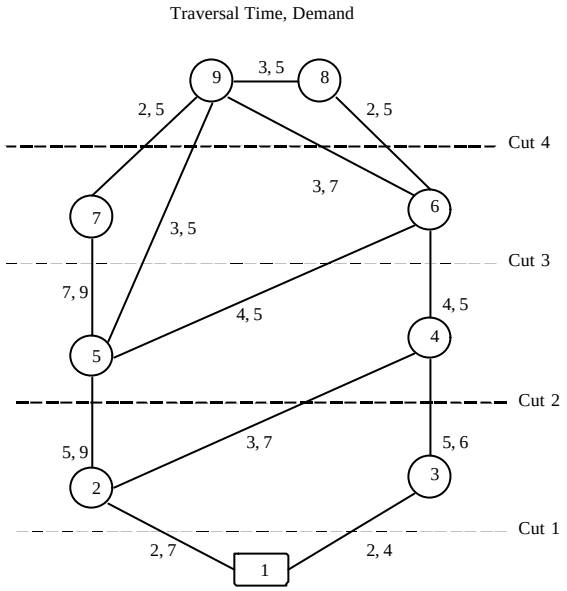


Figure 5.1 Taking Cuts as LB2

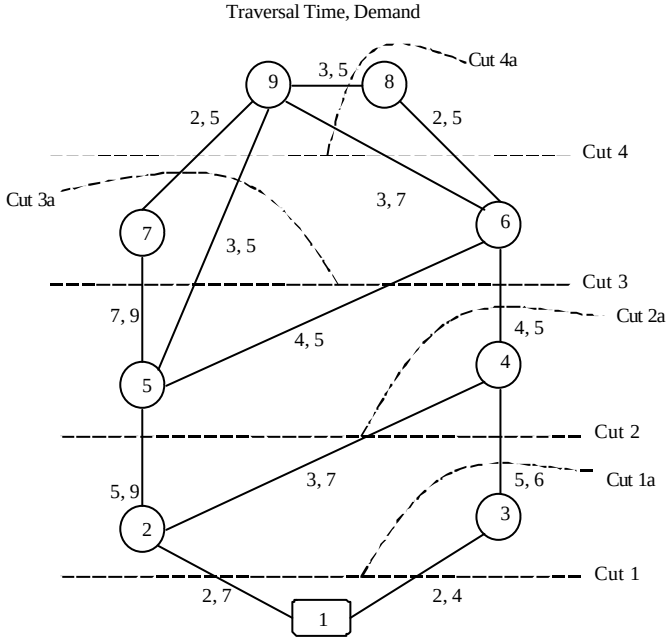


Figure 5.2 Taking Nodes one at a time, in increasing order of connectivity

The increasing order of degree selection procedure can be justified as follows:

The first cut in Fig 5.1. above is Cut(1-2, 1-3).

Consider the graph in Fig. 5.2. Initially, $U^* = \{2, 3\}$. If we chose arc (1,3) as the first node from U^* to be included in U then the cut would be Cut(1-2, 3-4). This cut has two arcs crossing the cut. If however, Node 2 was first included U then there would be three arcs crossing the cut. Taking either node first, the third cut regardless will be Cut(2-4, 2-5, 3-4). Using this simple reasoning, there is a better chance that including Node 3 in U first will provide a better second cut than including Node 2. This idea has been extended into a more general ruling that nodes will be included in U in non-decreasing order of degree.

Once all the nodes in U^* have been included in U and U^* is empty, then a new U^* must be calculated. U^* is calculated in the same way as it was calculated for LB2.

$U^* = \{i \in V \mid i \text{ is adjacent to a vertex in } U\}$.

$U = U \cup$ Node b, such that Degree[b] = Min(Degree[i]) where $i \in U^*$ and i is not already included in U.

For the example in Fig 5.2 above, the iterations while including nodes one at a time in increasing order of connectivity will produce sets U, U^* and V^* as follows:

Iteration Number	U	U^*	V^*
1	{1}	{2, 3}	{2, 3, 4, 5, 6, 7, 8, 9}
2	{1, 3}	{2}	{2, 4, 5, 6, 7, 8, 9}
3	{1, 3, 2}	{4, 5}	{4, 5, 6, 7, 8, 9}
4	{1, 3, 2, 4}	{5}	{5, 6, 7, 8, 9}
5	{1, 3, 2, 4, 5}	{6, 7}	{6, 7, 8, 9}
6	{1, 3, 2, 4, 5, 7}	{6}	{6, 8, 9}
7	{1, 3, 2, 4, 5, 7, 6}	{8, 9}	{8, 9}
8	{1, 3, 2, 4, 5, 7, 6, 8}	{9}	{9}

Table 5.1 The sets at each iteration

This procedure has an indirect affect on the procedure of altering L1. L1 will be re-calculated when all nodes in U^* are included in U and U^* is empty. L1 cannot be included in the bound until U^* is empty, since including L1 would permit the possibility of repeating an artificial edge which will be in the optimal solution. For example, if L1 was calculated in the example above by multiplying the number of vehicles by the shortest arc adjacent to the depot, arc (1,3), then arc (1,3) could also be in the current cut. If in the MCPM on H_s arc (1,3) was found as an artificial edge, and L1 included copies of that arc, then the number of copies of that arc included in the lower bound would be over estimated. The iterations of L1 has not changed from that of LB2.

5.1.1 Comments

The purpose of taking nodes one at a time was to examine more cuts in the graph. The purpose of examining more cuts in the graph is the have a better chance of finding a good cut, where a good cut is defined as a cut that will return a high MCPM on H_s. A high MCPM on Hs is more likely when there is fewer arcs in the cut. Including nodes one at a time gives us access to more cuts, and therefore a better chance of finding a good cut. The way in which H_s is constructed in LB2, copies of an arc which crosses the cut are made and not any arc close-by the cut. Thus, if the traversal time on arc (3,4) is higher than the traversal time on arc (1,3), and the number of vehicles having to cross the cut is the same, then the second cut may give a higher bound.

5.1.2 ANALYSIS OF LB(MOD):

The same graphs are used in this section as were used in chapter 4 to test the existing bounds. Since we concluded that LB2 was the best of the existing bounds, we will use LB2 as our comparison. The bound involving taking nodes one at a time will be referred to as LB(mod).

Time-Capacitated Problem

10 Arc Problem

Q_v	V	LB mod	LB 2
75	5	112	112
100	4	75	75
150	3	51	51
200	2	37	37

16 Arc Problem

Q_v	V	LB mod	LB 2
125	4	**105**	87
140	4	**95**	83
200	3	65	65
250	2	47	47

25 Arc Problem

Q_v	V	LB mod	LB 2
135	5	**149**	143
175	4	**115**	107
200	4	**99**	95
250	3	**83**	79

34 Arc Problem

Q_v	V	LB mod	LB 2
100	9	**245**	244
120	7	**179**	175
170	5	**120**	113
250	4	**94**	90

45 Arc Problem

Q_v	V	LB mod	LB 2
100	10	**357**	343
150	7	**239**	221
250	4	**133**	131
300	3	**107**	106

50 Arc Problem

Q_v	V	LB mod	LB 2
100	11	**337**	335
125	9	**261**	260
135	8	**227**	226
200	5	141	141

60 Arc Problem

Q_v	V	LB(mod)	LB 2
150	7	**190**	178
175	6	**158**	152
200	5	**131**	130
250	4	112	112

5.1.3 Conclusions

From the tables above, we can see that the new bound, LB(mod), makes a significant improvement to the lower bound over LB2. During testing we found that there was more significant improvements on the larger graphs. On smaller graphs, where fewer cuts will be made, it is less likely that there will be any improvement over LB2.

We also found that improvements to the bound tended to occur when more vehicles of lower capacity were being employed. LB(mod) finds more 'good' cuts and with more vehicles, more traversals are required across these cuts giving a better bound. It is useful to note that it is in these cases where there are more vehicles with less capacity that the branch and bound tends to run out of memory, since there are more possible permutations of routes if there are more vehicles.

5.2 ANALYSIS OF L1

In this section we discuss the use of L1 as an estimator of the number of traversals of arcs behind the cut, i.e. between the cut and the depot. The calculation of L1 is taken directly from the bound ZAW1 as written by Win[13]. No improvement has been made to this estimation of traversals behind the cut in any of the bounds discussed in this dissertation.

Though the information from the various previous cuts is carried through, the nodes of odd degree behind the cut are not matched. This means that LB2 is loosing some of the potential for estimating traversed arcs behind the cut. We know that all nodes of odd degree must be matched in order to form valid tours. But, LB2 does not match the odd nodes behind the cut. Therefore, the estimate of traversals due to the existence of odd nodes in the graph will be lower than if all odd nodes were matched.

However, it is a non-trivial task to formulate a bound which will incorporate the extra traversals needed to match odd nodes behind the cut, and beyond the time limitations of this dissertation. There is an interaction between the calculation of L1 and these odd nodes. Some of the traversals estimated by L1 may be between two nodes of odd degree, and in such case, including the odd nodes in the matching of H_s could result in over-estimating the traversals behind the cut.

5.3 LOWER BOUNDS AND THE BRANCH AND BOUND

As stated in chapter 2, the purpose of this thesis was to implement and review the existing lower bounds to the Capacitated Arc Routing Problem. The motivation for this was that improved bounds will improve the efficiency of the branch and bound algorithm as fewer sub-problems will be generated. This will lead to the ability of the algorithm to solve larger problems and also to reduce solution times.

As the branch and bound algorithm iterates, the lower bound gets updated. The lower bound is updated as sub-problems get fathomed. An improved lower bound will eliminate the sub-problems generated that have a lower MCPM than the lower bound. These sub-problems otherwise would have been permitted. This means that in order to reach the optimal solution, fewer sub-problems must be generated. The implications that this holds is that larger problems can be solved with improved lower bounds. As an added bonus, the branch and bound method will reach find optimal solutions quicker than before.

From the extensive testing of the branch and bound method carried out by Montwill and Naughton [11] in their dissertation, we know that the branch and bound is less likely to solve problems where more vehicles are being employed with a smaller capacity. This is due to the fact that more vehicles result in more postman paths being formed at each iteration, and consequently there are more possible permutations of arcs being serviced by particular paths. It is worthy of mention that it is in similar circumstances that LB(mod) has significantly better results over previous bounds.

CHAPTER 6

CONCLUSIONS AND FURTHER RESEARCH

6. 1 CONCLUSIONS

The purpose of this thesis was to implement existing lower bounding techniques and confirm that Bound LB2 by Benavent et al. supersedes other existing bounds. The motivation for this was that improved lower bounds will improve the efficiency of the branch and bound algorithm implemented by Montwill and Naughton.

We found that Bound LB2 did indeed perform better than any other bound, though it was better than Bound ZAW2 on only a few graphs. The reason why Bound LB2 is better than the other bounds is that it synthesises the best features of these other bounds. It includes the successive cut features of L1 and the traversal estimation strategy of LB1. During our testing, never did the Node Duplication Lower Bound beat LB2.

LB2 worked particularly well when vehicle capacity was low in relation to the weight on the arcs, and consequently there was a high number of vehicles required to service the graph. The improved bound performed well on the occassions when the branch and bound algorithm performed badly i.e. low capacity, high number of vehicles. This is when we most need good lower bounds.

There is much work yet to be done in the area of lower bounds to the CARP, as LB2 still fails to capture some of the characteristics of graphs, and consequently does not perform as well as it could. LB2 does not match the nodes of odd degree behind the cut, and therefore gives a poor estimate of the traversals required in that section of the graph. Also, LB2 only takes advantage of a single cut in any graph. If there are more than one ‘good cuts’ in the graph, LB2 takes the best of these cuts and returns the bound at that cut, and fails to capitalise on other cuts in the graph.

In any graph, there is a very large number of possible cuts, only a small number of which is evaluated by Bound LB2. To combat this, we developed a strategy to increment the cut by including a single node in U at a time, giving access to more cutsets. We selected nodes on the basis of their degree, and included them in U in increasing order of connectivity. This selection strategy improved the possibility of finding a 'good cut'.

We conclude from our analysis that LB2 supersedes all other existing bounds know by us to date and we further find that LB(mod) out-performs LB2. All bounds performed better on the sparse graphs taken from the Irish Rural road network than the well connected graphs provided by Hirabayashi et al.

6.2 THE WAY FORWARD

Following on from our work this summer, here are some areas which we feel are deserving of further study:

1. Construction of Routes beyond the cut

At each cut in the successive cutset bounds that use successive cutsets, the graph is split up into two or more components. Once the best cut is found, it would be useful to construct routes within the component(s) beyond the cut. These routes could then be fed into an Upper Bound Heuristic with the aim of getting an upper bound solution with routes that closely match the arcs used in the lower bound. Obviously, if the upper bound is close to the lower bound, it must be close to the optimal solution.

2. Using Heuristics to find 'Good Cuts'

It would be very beneficial if a method was devised to find 'good' cutsets in a graph. These cutsets could then be fed into the lower bound heuristic so that a matching can be carried out at the most advantageous cutsets in the graph. In this way, a higher lower bound can be found.

A possible method for finding such 'good cuts' would be to examine the routes formed by an upper bound heuristic. A good heuristic will offer routes which are close to an optimal solution and therefore will contain a similar set of extra traversals on the graph to those in the optimal solution. From the routes formed by the heuristic, we can evaluate which arcs in the graph are traversed most often. Then, by analysing the locations of these highly traversed arcs in the graph, we would be able to extract good cuts from the graph.

3. The Branching Strategy

In the algorithm devised by Montwill and Naughton, they tested various branching strategies including choosing the arc which, when included in the solution, returns the highest lower bound, and choosing the lowest cost arc. We suggest that changing the branching strategy to one which finds a valid postman tour as quick as possible would be more beneficial.

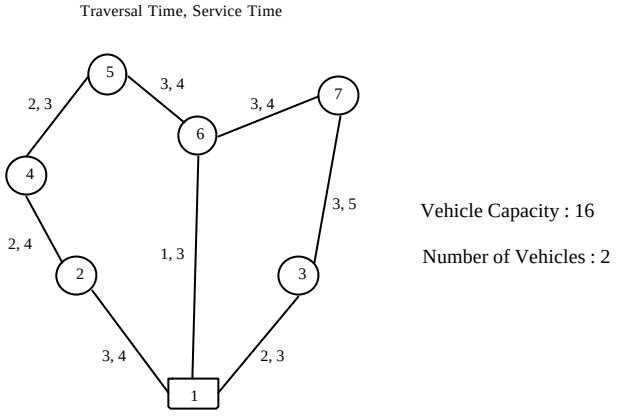


Figure 6.1 A new branching strategy

Suppose that the current solution offered by the branch and bound offered the following two routes for the graph in Fig. 6.1:

R_1 = (1-2-4-5-6-1)

R_2 = (1-3-7-6~1)

- Serviced Arc

~ Traversed Arc

The total time of R_1 is 18 and the total time of R_2 is 12. The length of R_1 exceeds the vehicle capacity of 16. If the servicing of arc (6,1) was swapped from R_1 to R_2, then we would have two valid postman paths. The new routes would be:

R_1 = (1-2-4-5-6~1)

R_2 = (1-3-7-6-1)

R_1 now takes 16 minutes while R_2 now takes 15 minutes, which is a valid postman tour. Using a branching strategy that would swap arcs between routes in this manner would find a postman tour very quickly. With the tight lower

bounds implemented in this dissertation and the tight upper bounds implemented by Keenan and Naughton, any valid postman tour would be very close to an optimal solution. From this postman tour which is close to the optimal solution, the branch and bound method should be able to solve more quickly and get an answer.

Finally, we feel that much work has yet to be done in the area of developing lower bounds. Following are some suggestions which may assist in improving the existing bounds:

4. Multi-cut Strategy

Keeping in mind the analysis of bounds made in chapters 4 and 5, the most improvement in existing bounds is to alter the bounds (in particular LB2) to take advantage of the existence of multiple good cutsets. This could take many forms. A possible method would be to store the last 'good' cut at each iteration. When you get to the end of the graph, go back to the last 'good' cut and do the matching of nodes of odd degree beyond that cut at this stage, and not before. The nodes of odd degree that do not lie on any of the cutsets may then be matched.

5. Improvement of L1

Finally, the area most obvious in need of attention is the improve the estimate of traversals between the cut and the depot. The calculation of L1, the current method of estimating these traversals, is quite simplistic. Matching of nodes of odd degree between the cut and depot needs to be incorporated into the lower bound in order to get a good bound on larger graphs.

REFERENCES

[**1**] Assad, A., Pearn, W., Golden, B.,

"The Capacitated Postman Problem: Lower Bounds and Solvable Cases"

*Am. J. Math. Management Sci. 7 (1, 2),*63-88, (1987)

[**2**] Ball, M. O., Derigs, U.,

"An analysis of alternative strategies for implementing matching algorithms"

Networks, 13, 225-261, (1988)

[**3**] Benavent, E., Campos, V., Corberan, A. and Mota, E.,

"The Capacitated Arc Routing Problem: Lower Bounds"

Networks, 22, 669-690, (1992)

[**4**] Derigs, U.,

"A Shortest Augmenting Path method for solving Minimal Perfect Matching Problems",

Networks, 18, 379-390, (1981)

[**5**] Eiselt, Gendreau, and Laporte,

"Arc Routing Problems, Part I: The Chinese Postman Problem",

Operations Research, 32(2): 231-242. (1995)

[**6**] Eiselt, Gendreau, and Laporte,

"Arc Routing Problems, Part II: The Rural Postman Problem",

Operations Research, 32(3): 399-414. (1994)

[**7**] Golden and Wong,

"Capacitated Arc Routing Problems",

Networks, 11, 305-315.

[**8**] Hirabayashi, R., Surawatari, Y., and Nishida, N,

"Node Duplication Lower Bounds for the Capacitated Arc Routing Problem",

Journal of the Operations Research Society of Japan. Vol. 35, 2, 119-133. (1992)

[**9**] Hirabayashi, R., Surawatari, Y., and Nishida, N,

"The Tour Construction Algorithm for the Capacitated Arc Routing Problem",

Asia - Pacific Journal of Operation Research 9, pp.155 -175, (1992).

[**10**] Keenan, P., Naughton, M.,

"Arc routing for rural Irish networks",

Presented at the 17th IFIP TC7 Conference on Systems Modelling and Optimisation, Prague. (10-14 July 1995).

[**11**] Montwill, P. and Naughton, M.

"The Capacitated Chinese Postman Problem"

M.Mangt.Sc. Dissertation, University College Dublin, (1994).

[**12**] Pearn, W.L.,

"New Lower Bounds for the Capacitated Arc Routing Problem",

Networks, 18, pp.181 -191, (1981)

[**13**] Win, Zaw,

"Contributions to routing Problems",

PhD Dissertation, University of Augsburg (1988).

APPENDIX A

ADJACENCY LISTS

The following are the adjacency lists of the graphs received from Hirabayshi et al. The files are listed in the following format:

- The first line contains the text 'No of Nodes'.
- The second line contains a number that represents the number of nodes in the graph.
- The third line contains the text 'No of Arcs'.
- The fourth line contains a number that represents the number of arcs in the graph.
- The fifth line contains the column headers
- Every subsequent line represents an arc in the graph. There are four columns in each line, the first is node i, the second, node j, the third is the demand on the arc, and the fourth holds the length of the arc. For the volume-capacitated graphs (those which start on the next page) whose names are of the format AA**, demand represents volume or load, and distance represents traversal time. For the time-capacitated graphs (following the volume-capacitated problems) whose names are of the format T**, demand represents service time, and distance represents traversal time.
- Starting at Node 1, all adjacent nodes are listed. Then all the adjacent nodes of Node 2 are listed, and so on until all nodes have had their adjacent nodes listed. Each arc in the graph is represented twice in the file, as each arc is adjacent to two nodes.

AA01

No of Nodes
8
No of Arcs
15

NodeI	NodeJ	Demand	Distance
1	8	28	707
1	7	13	399
1	6	8	798
1	5	57	954
2	8	12	721
2	5	15	985
2	4	14	496
2	3	65	530
3	8	21	1041
3	7	3	819
3	2	65	530
4	8	23	586
4	2	14	496
5	8	53	272
5	7	6	667
5	2	15	985
5	1	57	954
6	7	51	640
6	1	8	798
7	8	51	495
7	6	51	640
7	5	6	667
7	3	3	819
7	1	13	399
8	7	51	495
8	5	53	272
8	4	23	586
8	3	21	1041
8	2	12	721
8	1	28	707

AA02

No of Nodes
8
No of Arcs
15

NodeI	NodeJ	Demand	Distance
1	8	27	1082
1	3	40	737
2	7	30	370
2	6	60	780
2	5	5	745
2	4	54	166
2	3	25	510
3	8	23	696
3	7	50	311
3	4	7	663
3	2	25	510
3	1	40	737
4	8	55	1046
4	7	39	468
4	3	7	663
4	2	54	166
5	8	29	239
5	2	5	745
6	8	26	402
6	7	65	490
6	2	60	780
7	6	65	490
7	4	39	468
7	3	50	311
7	2	30	370
8	6	26	402
8	5	29	239
8	4	55	1046
8	3	23	696
8	1	27	1082

AA03

No of Nodes
8
No of Arcs
15

NodeI	NodeJ	Demand	Distance
1	7	25	665
1	6	54	550
1	4	26	702
1	3	50	344
1	2	9	212
2	8	51	187
2	7	34	858
2	5	59	574
2	4	43	846
2	1	9	212
3	7	54	626
3	6	46	762
3	4	50	509
3	1	50	344
4	7	5	300
4	6	0	742
4	3	50	509
4	2	43	846
4	1	26	702
5	2	59	574
6	7	6	490
6	4	0	742
6	3	46	762
6	1	54	550
7	6	6	490
7	4	5	300
7	3	54	626
7	2	34	858
7	1	25	665
8	2	51	187

AA05

No of Nodes

8

No of Arcs

15

NodeI	NodeJ	Demand	Distance
1	8	17	626
1	7	38	345
1	5	61	117
2	7	9	144
2	5	55	579
2	3	62	512
3	8	10	524
3	7	50	391
3	5	0	78
3	4	64	266
3	2	62	512
4	6	21	838
4	3	64	266
5	8	59	551
5	7	1	450
5	6	34	657
5	3	0	78
5	2	55	579
5	1	61	117
6	5	34	657
6	4	21	838
7	8	43	787
7	5	1	450
7	3	50	391
7	2	9	144
7	1	38	345
8	7	43	787
8	5	59	551
8	3	10	524
8	1	17	626

AA06

No of Nodes
8
No of Arcs
15

NodeI	NodeJ	Demand	Distance
1	8	6	361
1	6	15	36
1	4	54	304
1	3	57	510
2	8	15	732
2	6	3	358
2	4	46	531
3	8	37	527
3	7	8	589
3	5	8	721
3	4	12	792
3	1	57	510
4	8	37	430
4	7	34	262
4	6	52	339
4	3	12	792
4	2	46	531
4	1	54	304
5	7	41	226
5	3	8	721
6	4	52	339
6	2	3	358
6	1	15	36
7	5	41	226
7	4	34	262
7	3	8	589
8	4	37	430
8	3	37	527
8	2	15	732
8	1	6	361

AA08

No of Nodes
10
No of Arcs
15

NodeI	NodeJ	Demand	Distance
1	8	65	117
1	6	53	880
1	4	0	858
2	7	59	877
2	6	38	824
2	4	26	768
3	9	13	1040
3	4	36	803
4	9	43	316
4	3	36	803
4	2	26	768
4	1	0	858
5	9	44	903
5	8	32	490
6	8	21	771
6	7	50	728
6	2	38	824
6	1	53	880
7	9	54	148
7	6	50	728
7	2	59	877
8	10	40	626
8	6	21	771
8	5	32	490
8	1	65	117
9	7	54	148
9	5	44	903
9	4	43	316
9	3	13	1040
10	8	40	626

AA07

No of Nodes
10
No of Arcs
15

NodeI	NodeJ	Demand	Distance
1	7	35	863
1	3	66	604
2	10	23	323
2	9	1	519
2	6	40	160
3	8	3	520
3	1	66	604
4	7	46	811
4	5	40	616
5	10	9	182
5	6	35	510
5	4	40	616
6	9	59	386
6	8	49	563
6	5	35	510
6	2	40	160
7	9	20	224
7	8	36	827
7	4	46	811
7	1	35	863
8	10	16	215
8	7	36	827
8	6	49	563
8	3	3	520
9	7	20	224
9	6	59	386
9	2	1	519
10	8	16	215
10	5	9	182
10	2	23	323

AA08

No of Nodes
10
No of Arcs
15

NodeI	NodeJ	Demand	Distance
1	9	30	618
1	6	36	294
1	3	13	935
1	2	58	822
2	9	11	264
2	8	41	644
2	1	58	822
3	10	39	372
3	9	3	386
3	1	13	935
4	9	58	638
4	8	12	258
5	10	48	184
6	10	65	677
6	8	8	301
6	1	36	294
7	9	34	250
8	6	8	301
8	4	12	258
8	2	41	644
9	10	41	85
9	7	34	250
9	4	58	638
9	3	3	386
9	2	11	264
9	1	30	618
10	9	41	85
10	6	65	677
10	5	48	184
10	3	39	372

AA09

No of Nodes
6
No of Arcs
15

NodeI	NodeJ	Demand	Distance
1	6	40	582
1	5	46	426
1	4	26	173
1	3	33	488
1	2	27	232
2	6	59	351
2	5	53	583
2	4	1	244
2	3	25	691
2	1	27	232
3	6	62	1014
3	5	6	666
3	4	64	634
3	2	25	691
3	1	33	488
4	6	64	563
4	5	51	340
4	3	64	634
4	2	1	244
4	1	26	173
5	6	8	895
5	4	51	340
5	3	6	666
5	2	53	583
5	1	46	426
6	5	8	895
6	4	64	563
6	3	62	1014
6	2	59	351
6	1	40	582

AA10

No of Nodes
7
No of Arcs
15

NodeI	NodeJ	Demand	Distance
1	7	37	308
1	4	6	340
1	2	16	726
2	6	18	460
2	4	57	571
2	3	65	313
2	1	16	726
3	7	18	474
3	6	11	202
3	5	41	587
3	4	28	282
3	2	65	313
4	7	16	210
4	6	1	340
4	5	2	399
4	3	28	282
4	2	57	571
4	1	6	340
5	7	32	462
5	6	65	721
5	4	2	399
5	3	41	587
6	5	65	721
6	4	1	340
6	3	11	202
6	2	18	460
7	5	32	462
7	4	16	210
7	3	18	474
7	1	37	308

AA11

No of Nodes
7
No of Arcs
15

NodeI	NodeJ	Demand	Distance
1	7	20	397
1	3	12	357
1	2	35	530
2	7	32	601
2	6	42	205
2	4	61	162
2	3	22	311
2	1	35	530
3	7	33	291
3	6	20	481
3	5	9	595
3	4	35	302
3	2	22	311
3	1	12	357
4	7	48	572
4	3	35	302
4	2	61	162
5	7	33	827
5	6	19	560
5	3	9	595
6	7	46	764
6	5	19	560
6	3	20	481
6	2	42	205
7	6	46	764
7	5	33	827
7	4	48	572
7	3	33	291
7	2	32	601
7	1	20	397

AA12

No Nodes
7
No Arcs
15

NodeI	NodeJ	Demand	Distance
1	3	47	449
1	4	16	421
1	5	59	247
1	6	17	340
1	7	47	965
2	4	39	653
2	5	57	461
2	6	8	406
2	7	55	622
3	1	47	449
3	5	11	331
3	6	51	243
4	1	16	421
4	2	39	653
4	6	19	522
4	7	58	1150
5	1	59	247
5	2	57	461
5	3	11	331
5	7	18	770
6	1	17	340
6	2	8	406
6	3	51	243
6	4	19	522
6	7	4	654
7	1	47	965
7	2	55	622
7	4	58	1150
7	5	18	770
7	6	4	654

AA13

No Nodes
7
No Arcs
15

NodeI	NodeJ	Demand	Distance
1	3	42	844
1	7	8	383
2	3	53	371
2	4	13	333
2	5	13	290
2	6	10	396
2	7	53	307
3	1	42	844
3	2	53	371
3	4	61	186
3	5	24	389
3	6	57	763
3	7	26	624
4	2	13	333
4	3	61	186
4	6	47	682
4	7	43	503
5	2	13	290
5	3	24	389
5	6	16	597
5	7	3	583
6	2	10	396
6	3	57	763
6	4	47	682
6	5	16	597
7	1	8	383
7	2	53	307
7	3	26	624
7	4	43	503
7	5	3	583

AA14

No Nodes
7
No Arcs
15

NodeI	NodeJ	Demand	Distance
1	2	64	371
1	3	18	503
1	5	60	540
2	1	64	371
2	3	49	808
2	4	17	1297
2	5	38	910
2	6	34	585
2	7	12	307
3	1	18	503
3	2	49	808
3	5	49	349
3	6	7	604
3	7	39	502
4	2	17	1297
4	5	7	863
4	6	46	836
5	1	60	540
5	2	38	910
5	3	49	349
5	4	7	863
5	6	4	904
6	2	34	585
6	3	7	604
6	4	46	836
6	5	4	904
6	7	55	459
7	2	12	307
7	3	39	502
7	6	55	459

AA15

no nodes
8
no arcs
15

NodeI	NodeJ	Demand	Distance
1	4	42	873
1	5	58	640
1	7	34	444
2	3	20	583
2	7	53	740
3	2	20	583
3	4	63	471
3	5	16	293
3	6	65	441
4	1	42	873
4	3	63	471
4	6	50	866
4	8	37	318
5	1	58	640
5	3	16	293
5	6	42	334
5	7	1	466
6	3	65	441
6	4	50	866
6	5	42	334
6	7	51	602
6	8	22	563
7	1	34	444
7	2	53	740
7	5	1	466
7	6	51	602
7	8	40	357
8	4	37	318
8	6	22	563
8	7	40	357

AA16

no nodes
7
no arcs
15

NodeI	NodeJ	Demand	Distance
3	6	4	121
5	4	4	121
1	3	48	253
2	2	48	253
1	2	58	360
1	1	58	360
2	4	61	400
3	3	61	400
3	5	21	447
4	4	21	447
2	6	42	468
5	3	42	468
1	6	55	530
5	1	55	530
1	4	61	546
3	2	61	546
1	5	65	588
4	2	65	588
1	7	53	611
6	1	53	611
2	7	19	622
6	3	19	622
2	5	14	638
4	3	14	638
1	6	2	646
6	1	2	646
1	5	22	778
5	1	22	778
1	7	30	856
7	1	30	856

AA17

no nodes
8
no arcs
15

NodeI	NodeJ	Demand	Distance
1	2	40	242
1	3	35	620
1	4	19	488
1	5	22	459
1	7	27	672
2	1	40	242
2	4	12	669
2	8	6	430
3	1	35	620
3	4	39	269
3	5	35	623
3	6	38	440
4	1	19	488
4	2	12	669
4	3	39	269
4	6	56	668
4	8	29	861
5	1	22	459
5	3	35	623
5	6	32	980
5	7	61	555
5	8	27	854
6	3	38	440
6	4	56	668
6	5	32	980
7	1	27	672
7	5	61	555
8	2	6	430
8	4	29	861
8	5	27	854

AA18

no nodes
8
no arcs
15

NodeI	NodeJ	Demand	Distance
1	2	26	473
1	4	48	569
1	6	13	712
1	8	1	491
2	1	26	473
2	4	21	528
2	5	49	665
2	7	37	792
3	4	47	811
3	6	35	50
3	7	22	838
4	1	48	569
4	2	21	528
4	3	47	811
4	5	33	763
4	7	65	1027
4	8	14	262
5	2	49	665
5	4	33	763
5	7	14	300
6	1	13	712
6	3	35	50
6	8	18	997
7	2	37	792
7	3	22	838
7	4	65	1027
7	5	14	300
8	1	1	491
8	4	14	262
8	6	18	997

AA19

No Nodes
9
No Arcs
15

NodeI	NodeJ	Demand	Distance
1	2	11	514
1	3	52	490
1	6	4	447
1	9	16	330
2	1	11	514
2	3	4	132
2	4	4	696
2	5	35	440
3	1	52	490
3	2	4	132
3	5	38	311
3	7	1	747
3	8	43	814
3	9	30	815
4	2	4	696
4	7	45	924
4	8	21	958
5	2	35	440
5	3	38	311
5	6	19	132
6	1	4	447
6	5	19	132
6	9	66	775
7	3	1	747
7	4	45	924
8	3	43	814
8	4	21	958
9	1	16	330
9	3	30	815
9	6	66	775

AA20

no nodes
9
no arcs
15

NodeI	NodeJ	Demand	Distance
1	5	19	570
1	9	63	318
2	7	3	759
2	8	21	370
3	4	27	354
3	5	46	215
3	7	54	487
4	3	27	354
4	5	20	510
4	7	2	654
4	9	33	520
5	1	19	570
5	3	46	215
5	4	20	510
5	8	20	818
6	7	33	1042
6	8	2	547
7	2	3	759
7	3	54	487
7	4	2	654
7	6	33	1042
7	9	54	1174
8	2	21	370
8	5	20	818
8	6	2	547
8	9	47	630
9	1	63	318
9	4	33	520
9	7	54	1174
9	8	47	630

AA21

No Nodes
9
No Arcs
15

NodeI	NodeJ	Demand	Distance
1	2	55	536
1	4	17	149
1	5	38	921
1	7	20	553
1	9	9	672
2	1	55	536
2	9	22	164
3	7	55	875
3	9	40	864
4	1	17	149
4	5	31	972
4	8	29	301
5	1	38	921
5	4	31	972
5	7	4	441
6	8	32	1185
6	9	33	280
7	1	20	553
7	3	55	875
7	5	4	441
7	9	50	161
8	4	29	301
8	5	32	1185
8	9	44	927
9	1	9	672
9	2	22	164
9	3	40	864
9	5	33	280
9	7	50	161
9	8	44	927

T10

No Nodes
7
No Arcs
10

NodeI	NodeJ	Demand	Distance
1	3	20	7
1	2	50	20
2	4	40	15
2	1	50	20
3	6	8	5
3	4	25	15
3	1	20	7
4	7	50	20
4	6	50	20
4	5	27	10
4	3	25	15
4	2	40	15
5	7	24	8
5	4	27	10
6	7	19	11
6	4	50	20
6	3	8	5
7	6	19	11
7	5	24	8
7	4	50	20

T16

No Nodes
11
No Arcs
16

NodeI	NodeJ	Demand	Distance
1	5	20	9
1	4	28	13
1	2	25	10
2	6	13	4
2	3	16	8
2	1	25	10
3	6	40	25
3	2	16	8
4	7	30	13
4	5	30	14
4	1	28	13
5	7	30	12
5	6	28	11
5	4	30	14
5	1	20	9
6	8	24	10
6	5	28	11
6	3	40	25
6	2	13	4
7	9	28	14
7	8	20	7
7	5	30	12
7	4	30	13
8	11	45	20
8	10	36	17
8	7	20	7
8	6	24	10
9	10	29	18
9	7	28	14
10	9	29	18
10	8	36	17
11	8	45	20

T25

No Nodes
16
No Arcs
25

NodeI	NodeJ	Demand	Distance
1	4	27	11
1	2	22	8
2	5	10	9
2	3	16	6
2	1	22	8
3	5	30	10
3	2	16	6
4	10	30	8
4	5	22	6
4	1	27	11
5	7	23	11
5	6	19	9
5	4	22	6
5	3	30	10
5	2	10	9
6	9	14	7
6	8	19	7
6	5	19	9
7	11	13	4
7	8	14	3
7	5	23	11
8	12	17	9
8	9	29	13
8	7	14	3
8	6	19	7
9	8	29	13
9	6	14	7
10	15	40	15
10	14	13	4
10	11	28	14
10	4	30	8
11	15	50	25
11	12	25	10
11	10	28	14
11	7	13	4
12	16	37	19
12	13	20	8
12	11	25	10
12	8	17	9
13	16	26	6
13	12	20	8
14	15	20	10
14	10	13	4
15	16	40	13

15	14	20	10
15	11	50	25
15	10	40	15
16	15	40	13
16	13	26	6
16	12	37	19

T34

No Nodes
18
No Arcs
34

NodeI	NodeJ	Demand	Distance
1	6	20	9
1	5	27	11
1	2	22	8
2	7	40	15
2	6	50	25
2	3	16	6
2	1	22	8
3	7	30	10
3	4	42	17
3	2	16	6
4	3	42	17
5	9	22	8
5	8	19	7
5	6	22	6
5	1	27	11
6	9	25	11
6	7	19	9
6	5	22	6
6	2	50	25
6	1	20	9
7	10	13	5
7	6	19	9
7	3	30	10
7	2	40	15
8	12	21	8
8	9	19	10
8	5	19	7
9	13	9	3
9	10	22	11
9	8	19	10
9	6	25	11
9	5	22	8
10	14	13	4
10	11	14	7
10	9	22	11
10	7	13	5
11	14	13	7
11	16	24	8
11	10	14	7
12	17	27	14
12	15	23	10
12	13	30	19
12	8	21	8
13	15	22	11

13	14	29	18
13	12	30	19
13	9	9	3
14	15	15	7
14	16	17	9
14	11	13	7
14	13	29	18
14	10	13	4
15	18	18	6
15	17	20	10
15	16	15	6
15	14	15	7
15	13	22	11
15	12	23	10
16	18	13	7
16	15	15	6
16	14	17	9
16	11	24	8
17	18	25	15
17	15	20	10
17	12	27	14
18	17	25	15
18	16	13	7
18	15	18	6

T45

No Nodes
28
No Arcs
45

NodeI	NodeJ	Demand	Distance
1	6	20	10
1	2	14	6
2	7	13	6
2	3	19	8
2	1	14	6
3	8	14	7
3	4	10	4
3	2	19	8
4	8	15	10
4	5	11	5
4	3	10	4
5	4	11	5
6	9	19	11
6	7	45	20
6	1	20	10
7	10	27	13
7	8	14	5
7	6	45	20
7	2	13	6
8	12	10	4
8	7	14	5
8	4	15	10
8	3	14	7
9	19	19	8
9	15	28	13
9	6	19	11
10	16	30	14
10	15	16	9
10	11	22	10
10	7	27	13
11	17	14	9
11	13	29	10
11	10	22	10
12	14	11	4
12	13	9	3
12	8	10	4
13	18	20	10
13	14	9	5
13	12	9	3
13	11	29	10
14	13	9	5
14	12	11	4
15	22	19	8
15	19	29	10

15	16	16	8
15	10	16	9
15	9	28	13
16	22	22	10
16	17	15	5
16	15	16	8
16	10	30	14
17	23	22	10
17	16	15	5
17	11	14	9
18	24	30	15
18	23	28	11
18	13	20	10
19	22	19	8
19	21	18	7
19	20	17	6
19	15	29	10
19	9	19	8
20	25	13	6
20	21	9	3
20	19	17	6
21	26	18	7
21	22	13	4
21	20	9	3
21	19	18	7
22	27	40	15
22	23	15	4
22	21	13	4
22	19	19	8
22	16	22	10
22	15	19	8
23	28	29	13
23	24	18	7
23	22	15	4
23	18	28	11
23	17	22	10
24	28	25	8
24	23	18	7
24	18	30	15
25	26	20	10
25	20	13	6
26	25	20	10
26	21	18	7
27	22	40	15
28	24	25	8
28	23	29	13

T50

No Nodes
28
No Arcs
50

NodeI	NodeJ	Demand	Distance
1	6	20	10
1	2	14	6
2	7	13	6
2	3	19	8
2	1	14	6
3	8	14	7
3	4	10	4
3	2	19	8
4	8	15	10
4	5	11	5
4	3	10	4
5	12	30	13
5	4	11	5
6	10	14	8
6	9	19	11
6	7	45	20
6	1	20	10
7	10	27	13
7	8	14	5
7	6	45	20
7	2	13	6
8	12	10	4
8	7	14	5
8	4	15	10
8	3	14	7
9	19	19	8
9	15	28	13
9	6	19	11
10	6	14	8
10	16	30	14
10	15	16	9
10	11	22	10
10	7	27	13
11	17	14	9
11	13	29	10
11	10	22	10
12	5	30	13
12	14	11	4
12	13	9	3
12	8	10	4
13	18	20	10
13	14	9	5
13	12	9	3
13	11	29	10

14	13	9	5
14	12	11	4
15	22	19	8
15	19	29	10
15	16	16	8
15	10	16	9
15	9	28	13
16	23	25	11
16	22	22	10
16	17	15	5
16	15	16	8
16	10	30	14
17	23	22	10
17	16	15	5
17	11	14	9
18	24	30	15
18	23	28	11
18	13	20	10
19	22	19	8
19	21	18	7
19	20	17	6
19	15	29	10
19	9	19	8
20	25	13	6
20	21	9	3
20	19	17	6
21	25	13	6
21	26	18	7
21	22	13	4
21	20	9	3
21	19	18	7
22	27	40	15
22	23	15	4
22	21	13	4
22	19	19	8
22	16	22	10
22	15	19	8
23	16	25	11
23	28	29	13
23	24	18	7
23	22	15	4
23	18	28	11
23	17	22	10
24	28	25	8
24	23	18	7
24	18	30	15
25	21	13	6
25	26	20	10
25	20	13	6
26	25	20	10
26	21	18	7

27	28	13	4
27	22	40	15
28	27	13	4
28	24	25	8
28	23	29	13

T60

No Nodes
38
No Arcs
60

NodeI	NodeJ	Demand	Distance
1	7	22	7
1	5	16	5
1	2	10	4
2	5	12	6
2	3	20	8
2	1	10	4
3	6	13	5
3	4	12	6
3	2	20	8
4	9	20	7
4	3	12	6
5	8	13	5
5	7	13	5
5	2	12	6
5	1	16	5
6	9	13	5
6	8	14	7
6	3	13	5
7	11	9	4
7	5	13	5
7	1	22	7
8	13	25	10
8	12	9	4
8	6	14	7
8	5	13	5
9	13	13	4
9	10	20	10
9	6	13	5
9	4	20	7
10	18	20	10
10	14	13	6
10	9	20	10
11	15	12	5
11	12	9	4
11	7	9	4
12	17	20	8
12	16	20	7
12	13	25	10
12	11	9	4
12	8	9	4
13	18	13	4
13	17	13	4
13	12	25	10
13	9	13	4

13	8	25	10
14	21	9	3
14	20	9	3
14	10	13	6
15	24	20	10
15	16	6	2
15	11	12	5
16	22	20	7
16	15	6	2
16	12	20	7
17	23	13	4
17	13	13	4
17	12	20	8
18	27	20	10
18	23	16	5
18	19	18	6
18	13	13	4
18	10	20	10
19	29	19	11
19	28	20	10
19	18	18	6
20	21	9	3
20	14	9	3
21	20	9	3
21	14	9	3
22	25	10	3
22	23	11	9
22	16	20	7
23	26	10	3
23	22	11	9
23	18	16	5
23	17	13	4
24	31	13	4
24	25	13	4
24	15	20	10
25	32	13	4
25	31	13	4
25	26	13	4
25	24	13	4
25	22	10	3
26	27	10	3
26	25	13	4
26	23	10	3
27	28	10	3
27	26	10	3
27	18	20	10
28	33	40	20
28	32	20	11
28	27	10	3
28	19	20	10
29	33	27	19

29	30	13	4
29	19	19	11
30	38	23	9
30	37	27	11
30	29	13	4
31	34	10	6
31	25	13	4
31	24	13	4
32	35	20	10
32	34	24	8
32	33	26	13
32	28	20	11
32	25	13	4
33	36	18	7
33	32	26	13
33	29	27	19
33	28	40	20
34	32	24	8
34	31	10	6
35	36	19	9
35	32	20	10
36	35	19	9
36	33	18	7
37	30	27	11
38	30	23	9